NISTIR 5367

Prediction of Geometric-Thermal Machine Tool Errors by Artificial Neural Networks

D. E. Gilsinn
M. A. Donmez

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Gaithersburg, MD 20899

Supported by:

Navy Manufacturing Technology
(MANTECH) Program

Monitoring Office:

NAVAL Industrial Resources Support
Activity (NAVIRSA)
Attn: Leo Plonsky (203)

NIST

NISTIR 5367

Prediction of Geometric-Thermal Machine Tool Errors by Artificial Neural Networks

D. E. Gilsinn
M. A. Donmez

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Gaithersburg, MD 20899

Supported by:

Navy Manufacturing Technology
(MANTECH) Program

Monitoring Office:

NAVAL Industrial Resources Support
Activity (NAVIRSA)
Attn: Leo Plonsky (203)

April 1994

U.S. DEPARTMENT OF COMMERCE
Ronald H. Brown, Secretary

TECHNOLOGY ADMINISTRATION
Mary L. Good, Under Secretary for Technology

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
Arati Prabhakar, Director

Abstract

In machining operations, the precision of the workpiece dimensions depends on the accuracy of the relative position of the cutting tool and the workpiece. Among the key factors that affect the accuracy of this relative position are the geometric errors of the machine tool and the thermal effects on these geometric errors. Recent work on developing models to predict volumetric errors on NC lathes has led to a synthesis technique the combines the modeling of individual axis related geometric-thermal components by way of a rigid body kinematic model to produce predicted errors in the work volume of the machine. Homogeneous coordinate transformations are the tools used to synthesize the individual error component models into the unified volumetric error model. The individual error components are often modeled as polynomial functions of a component's slide position and the temperature state of the machine, which is typically measured by thermal sensors located on the machine tool. An alternative method of modeling these component errors is described in this report. Neural network computing is shown to be a viable technique for developing mappings between machine tool component error measurements and the vector consisting of both a component slide position and the temperature state of the machine as reported by the thermal sensors. The conjugate gradient algorithm, used to compute the optimum neural network weights for the machine tool error components, is described. A case study of the mapping results for one component error of an actual NC lathe is given. Finally, the source codes for the neural network algorithm and the conjugate gradient algorithm are given in FORTRAN.

Keywords: conjugate gradient algorithm; gradient descent algorithm; machine tool error compensation; machine tool errors; neural networks; nonlinear optimization

Acknowledgement

We wish to thank Steve Osella for his careful review of the document. His suggestions helped clarify serveral points in the text. We also wish to thank Judy Barnard for her exceptional professionalism in the preparation of the document.

TABLE OF CONTENTS

1.0 Introduction 1

2.0 Machine Tool Error Compensation 3

3.0 Kinematic Model 3
- 3.1 General Model Structure 4
- 3.2 Predictive Machine Calibration 6
- 3.3 Individual Error Contributions 7
  - 3.3.1 Linear Displacement Errors 7
  - 3.3.2 Angular Errors 9
  - 3.3.3 Straightness-Parallelism-Orthogonality 9
  - 3.3.4 Spindle Thermal Drift 9

4.0 Some Regression Approach Shortcomings 10

5.0 Neural Networks 11
- 5.1 Parallel Distributed Processing Model 11
  - 5.1.1 Processing Units 12
  - 5.1.2 State of Activation 12
  - 5.1.3 Output Function 13
  - 5.1.4 Connectivity 13
  - 5.1.5 Rule of Propagation 14
  - 5.1.6 Activation Rule 18
  - 5.1.7 Connectivity Modification 18
  - 5.1.8 The Usage Environment 18
- 5.2 Learning Paradigm 19

6.0 Gradient Descent Algorithms 19
- 6.1 Algorithm Terminology 20
- 6.2 Computing the Gradient 23
- 6.3 Descent Strategy 28
- 6.4 Back Propagation Algorithm 29
- 6.5 Scaled Conjugate Gradient Algorithm 30

7.0 Adjusting for Ill Conditioning 38

8.0 Statistical Goodness-of-Fit Measures 42

9.0 Geometric-Thermal Error Component Mapping 44
9.1 X-Displacement (Down) 45
9.2 X-Displacement (Up) 59
9.3 X-Displacement (Down) with X-Axis Related Thermocouples 63
9.4 X-Displacement (Down) with Z-Axis Related Thermocouples 63
9.5 X-Displacement (Up) with X-Axis Related Thermocouples 67
9.6 X-Displacement (Up) with Z-Axis Related Thermocouples 74
9.7 X-Displacement (Down) with 50 Hidden Nodes 74
9.8 X-Displacement (Down) with 40 Hidden Nodes 80

10. Conclusions 80

11. Bibliography 88

Appendix A: Main Program Listing 91

Appendix B: Sample Output File 130

Appendix C: Sample Input File 140

LIST OF FIGURES

Figure 1. The component errors that contribute to the combined X and Z errors. 8

Figure 2. Perceptron model of a neural network processing unit. 15

Figure 3. A sample of a fully connected feed-forward network with three input nodes, four hidden nodes, and two output nodes 16

Figure 4. Schematic of the optimization strategy. 21

Figure 5. Turning center with thermocouple locations identified 46

Figure 6. Histogram plot of the residual error distribution after the fit of the X-displacement (down) data 51

Figure 7. Normality plot for residuals of the X-displacement (down) data. 52

Figure 8. Correlation Coefficient Plot for the residuals of the X-displacement (down) data 53

Figure 9. Histogram plot of the residual error distribution after the test of the X-displacement (down) data. All data columns used. 55

Figure 10. Error reduction curve for training on data with failed thermocouples removed. 56

Figure 11. Histogram of residuals for training data with failed thermocouples removed 57

Figure 11A. Histogram of residuals for testing the trained neural net with failed thermocouples removed 58

Figure 12. Regularized RMS error training curve for X-displacement upwards with failed thermocouples removed. 60

Figure 13. Histogram of residuals after neural net training on X-displacement upward data with failed thermocouples removed 61

Figure 13A. Histogram of residuals after test data applied to neural net for X-displacement, upwards. Failed thermocouples removed. 62

Figure 14. Regularized RMS error training curve for fitting X-displacement downwards data against X-axis thermocouples 64

Figure 15. Histogram of residuals after training with X-displacement downwards data for X-axis related thermocouples. 65

Figure 15A. Histogram of residuals for testing the trained neural net with X-displacement downward data against X-related thermocouples 66

Figure 16. Regularized RMS error training curve for X-displacement downwards data against Z-axis related thermocouples. 68

Figure 17. Histogram of residuals after training of X-displacement downwards data against Z-axis related thermocouples 69

Figure 17A. Histogram of residuals for test data for X-displacement downwards against Z-axis 70

Figure 18. Regularized RMS error training curve for X-displacement upward data against X-axis related thermocouples. 71

Figure 19. Histogram of residuals for training data of X-displacement upwards data against X-axis related thermocouples 72

Figure 19A. Histogram of residuals of test data of X-displacement upwards against X-axis related thermocouples 73

Figure 20. Regularized RMS error training curve for X-displacement upwards data against Z-axis related thermocouples 75

Figure 21. Histogram of residual errors for training of X-displacement upwards data against Z-axis related thermocouples 76

Figure 21A. Histogram of residual errors for testing X-displacement upwards data against Z-axis related thermocouples 77

Figure 22. Regularized RMS error curve for training X-displacement downwards data against all active thermocouples using 50 hidden nodes. 78

Figure 23. Histogram of residual errors for training X-displacement downwards data against all active thermocouples using 79

Figure 23A. Histogram of residual errors for testing X-displacement downwards data against all active thermocouples using 50 hidden nodes. 81

Figure 24. Regularized RMS error curve for training X-displacement downwards data against all active thermocouples using 40 hidden nodes. 82

Figure 25. Histogram of residuals for training X-displacement downwards data against all active thermocouples using 40 hidden nodes. 83

Figure 25A. Histogram of residuals for testing X-displacement downwards data against all active thermocouples using 40 hidden nodes. 84

1.0 Introduction

In machining operations, the precision of the workpiece dimensions depends on the accuracy of the relative position of the cutting tool and the workpiece. Among the key factors that affect the accuracy of this relative position are the geometric errors of the machine tool and the thermal effects on these geometric errors. Geometric errors are caused by the unwanted motions of machine elements such as carriages, cross-slides, and work-tables. These motions occur because of geometric imperfections and misalignments of the machine tool design. Heat generated by the machine tool and the cutting operation causes temperature changes of the machine tool elements and environment. Due to the complex geometry of the machine structure, concentrated heat sources such as the drive motors and spindle bearings, create thermal gradients along the machine structure. Spindle growth, lead screw expansion, and a significant part of the machine structure deformation are the results of these temperature changes and gradients; therefore, reducing the geometric and thermally-induced errors of a machine tool is a key requirement for improving the workpiece accuracy. In addition to the improved workpiece accuracy, productivity is increased by the elimination of the nonproductive warm-up cycle of 1-2 hours commonly used to reach thermal equilibrium when machining precision parts.

Recent work by Donmez and his colleagues [1,2,3], on developing models to predict volumetric errors of NC lathes, has led to a synthesis technique that combines the modeling of individual axis related geometric-thermal error components by way of a rigid body kinematic model to produce predicted errors in the work volume of the machine. Wu [4] describes several other error component modeling techniques. For example, characterizing a 3-axis machine tool requires 21 geometric error components plus spindle errors. Some of these error components are purely temperature related and others are functions of both temperature and nominal position. For a formal definition of the various machine tool error components see Hocken [20].

Homogeneous coordinate transformations under rigid body kinematic assumptions have been used by Donmez [1,5] to synthesize individual error component models into a unified volumetric error model. The rigid body kinematic assumption implies that the machine component errors at each slide depend on their own coordinate system and are not affected by the movements of other slides. That is, measurements on each slide are independent with regard to position on other slides. The end result of the synthesis produces volumetric components as functions of several individual error

components and nominal tool positions.

Individual error components are often modeled as polynomial functions of the slide position and temperature state of the machine, which is typically measured by thermal sensors located on the machine tool. For a survey of these measurement techniques see Wu [4]. For a specific application to an NC lathe see Donmez [5].

In this report we consider an alternative method of modeling the individual error components. Instead of polynomial functions of axis position and selected temperature measurements we consider the use of neural networks as a technique of developing mappings between patterns of recorded position and machine thermal states and measured component errors. An object of this modeling is to determine, if possible, limits on the number of data samples required and how many and where thermal sensors should be placed. This report, however, concentrates on the documentation of a particular neural net program and its application to sample data sets.

Neural network computing is a computational method that uses networks of simple processing elements or neural nodes to implement a desired mapping relationship between input and output data patterns. A neural node is a computational device that takes a number of inputs from either external sources or other nodes, performs sums of weighted inner products of these inputs, applies a fixed nonlinear transformation to this inner product and passes the result on to the next layer of neural nodes or to external outputs. Although neural networks have their early history connected with biological questions of how the human brain works, certain forms of neural networks, called multilayer feedforward networks, can be shown to produce functions that will uniformly approximate continuous functions of many variables in the sense of mathematical approximation theory (Cybenko [13]). For a discussion of the biological background of neural networks see Ritter, Martinetz and Schulten [6]. For a discussion of the general mathematical problems of neural computing see Cybenko [7].

The neural networks computer program used in this study is a modified version of one develped by Dr. James L. Blue at NIST for a study of optical character recognition (see Grother and Blue [8]). The optimization portion of the program is based on the scaled conjugate gradient algorithm of Møller [9]. Stefan Leigh of the Statistical Engineering Division at NIST developed the DATAPLOT output described in section 9. The authors also wish to acknowledge John Meyer,

Phil Nanzetta and Don Blomquist for their continued support.

2.0 Machine Tool Error Compensation

Error compensation is defined as a method of cancelling the effects of systematic errors either by directly or indirectly measuring these errors, or by predicting them using a model previously established for the process. For a discussion of the techniques involved in machine tool error compensation see Donmez [5, 21], Zhang [10]. Zhang describes geometric error corrections for coordinate measuring machines whereas Donmez details the application of geometric-thermal error correction to an NC lathe. The errors for which the system compensates are quasistatic geometric and thermally-induced errors. These errors are predicted by using relationships established between systematic errors and particular machine tool temperature profiles. In the process of building these relationships, the geometric errors are measured using, for example, laser interferometry and high precision capacitance probes. The temperature profile is determined by monitoring thermocouples placed in critical locations. Some of these locations are the leadscrew nuts and bearing housings, the spindle bearing housings, the headstock, the bed, several points on the cross slide and the carriage bodies.

At present the relationships for each type of error, such as linear displacement, straightness, yaw, and orthogonality, are established by applying least-squares curve fitting techniques to the error data and the corresponding temperature profiles for each element of the machine. After predicting each error component, the compensation system uses the principles of rigid body kinematics to combine the error components in order to find the error at the cutting tool.

The actual error compensation is achieved by sending the error values to the machine controller. The machine tool controller acts on the compensation values in software, thus, no modification to the machine control hardware is necessary.

3.0 Kinematic Model

The rigid body kinematic model referred to in section 2.0 is a general mathematical model that calculates the vector error at the cutting tool from a large number of reproducible error components. These components correspond to errors contributed by machine structure elements.

The individual errors are decomposed into their geometric and thermally-induced error components.

3.1 General Model Structure

The mathematical model used to predict volumetric errors, is generated by using homogeneous transformation matrix manipulations, with the assumptions of rigid body kinematics. These transformations describe the spatial relationships between the machine tool structural elements. A homogeneous coordinate transformation matrix in three-dimensional space is a 4 X 4 matrix. It is used to express a homogeneous coordinate vector in one coordinate system with respect to another coordinate system. Similarly, a homogeneous transformation matrix can be used to represent one coordinate system with respect to another or reference coordinate system. If the coordinate frame is embedded in an object, then a homogeneous matrix describes the relative position and orientation of this object with respect to another object or coordinate frame in space. An important feature of homogeneous coordinate transformations is that they can be multiplied in series to describe the position and orientation of one object with respect to several coordinate frames.

The homogeneous transformation used to describe the spatial orientation and location of a machine structural element takes the form

$$T = \begin{bmatrix} o_{1x} & o_{2x} & o_{3x} & p_x \\ o_{1y} & o_{2y} & o_{3y} & p_y \\ o_{1z} & o_{2z} & o_{3z} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

where the vectors $\mathbf{o}_1$, $\mathbf{o}_2$, $\mathbf{o}_3$ describe the axis system orientation of a coordinate frame with respect to another coordinate frame. The vector p is the position vector of the origin of the first coordinate frame with respect to the second.

Since a machine tool can be considered as a chain of linkages, an approach that describes the spatial geometry of linkages with respect to a reference frame by matrix multiplications can be used to determine the spatial relationship between the cutting tool and the workpiece. Thus, by

multiplying the homogeneous transformation matrices corresponding to a series of elements such as the carriage, cross-slide, cutting tool, workpiece and spindle, the cutting tool and the workpiece position can be described with respect to a conveniently chosen fixed reference frame (Donmez [1]). In ideal conditions, the cutting tool follows the contours of the ideal workpiece geometry. Thus, at any time during the operation, the resultant homogeneous transformation matrices,

$$T_{TOOL} \tag{2}$$

for the cutting tool and

$$T_{WORK} \tag{3}$$

for the point on the workpiece, should be identical. However, due to the individual errors involved, these two matrices are not identical in reality. The total error, E, is represented by the following equation:

$$T_{WORK}=T_{TOOL}E \tag{4}$$

The equation

$$E=T_{TOOL}^{-1}T_{WORK} \tag{5}$$

gives the resultant error matrix.

As discussed earlier homogeneous transformations can be multiplied in order to locate a structural frame with respect to a reference frame such as, for example, the machine coordinate system. On a lathe, for example, the final tool transformation may be a combination of transformations starting with the horizontal slide (z), then the vertical slide (x), followed by the turret and finally the tool. Thus one has the string of multiplications

$$T_{TOOL} = T_Z\ {}^{Z}T_X\ {}^{X}T_{TURRET}\ {}^{TURRET}T_{TOOL} \tag{6}$$

where the leading superscripts represent the most immediate reference frame. Thus the second transformation describes the orientation of the x-slide system with respect to the z-slide system. The z-slide is ultimately referenced to the machine coordinate system. A similar set of component matrices combine to form the workpiece transformation. This might, for example, be a combination of the spindle coordinate system transformation followed by a workpiece transformation with

respect to the spindle.

Many of the components of these matrices are small so that the matrix multiplications make use of the algebra of infinitesimals. That is, higher order terms are ignored. However, each of the matrices are composed of elements that are functions of measured quantities. The individual quantities will be described below.

3.2 Predictive Machine Calibration

The error vector at the tool tip consists of individual error components corresponding to different structural elements of the machine tool. This is the result of the product of the homogeneous transformations. In order to predict the resultant volumetric error at any location and at any time in the machine work zone, all of these individual error components must be predicted. Although these components are geometric error components of the structural elements of the machine tool, their characteristics change as a result of such factors as thermal effects and loading conditions. Currently, only the effects of nominal position and the thermal state of the machine on the individual error components are considered. Factors such as tool post and workpiece deflection are not considered although the mathematical model can account for such items. Thus, any error (e) can be considered as some combination of nominal position (x) and temperature. This may be expressed as:

$$e=a_0+a_1x+a_2x^2+\cdots+b_1T+b_2T^2+\cdots \tag{7}$$

The coefficients in this equation can be determined by using least-squares curve-fitting techniques. The data used in this analysis is obtained by making error measurements while monitoring axes positions and temperatures on the machine structure. A description of this methodology and related uncertainties due to data fitting for a particular machine tool are given in Donmez [5]. The need to only measure individual axis related error components is at the basis of the rigid body kinematic assumption.

Since a coordinate frame is assigned to each element of the machine tool, and these elements are designed for shear rather than for bending, it is possible to determine the error at any nominal position in a work zone by measuring the errors along the orthogonal axes of the work zone. This procedure decreases the number of measurement points significantly since it eliminates the need for grid measurements in the whole work zone.

3.3 Individual Error Contributions

The error components are classified into four groups with similar characteristics, measurement procedures and measurement instrumentation. The four groups are: 1) linear displacement errors, 2) angular errors, 3) straightness-parallelism-orthogonality errors and 4) spindle thermal drift. A schematic of all of the contributing errors is shown in Figure 1. This shows the nine axis related errors that contribute to the total work volume error of an NC turning center.

3.3.1 Linear Displacement Errors

Linear displacement errors are errors of position of machine elements along their axes of motion. This type of error is the direct result of drive system errors, such as erroneous lead of the ballscrew, ballscrew misalignment, and coupling errors between the ballscrew and the feedback system. In addition, displacement errors include displacements due to thermal effects on the ballscrew. Since thermal effects are correlated with the effective length of the ballscrew, the linear displacement errors are not easily decoupled and include position and thermally-induced components. For example, it has been found (see Gilsinn, et. al. [11]) that the linear displacement error (δ) for a specific NC lathe at a nominal position (x) as a function of ballscrew bearing housing temperature (T) can be described by an equation of the form:

$$\delta = a_0 + a_1 x + a_2 x^2 + b_1 T + c_1 x T. \tag{8}$$

where T here represents a particular thermocouple.

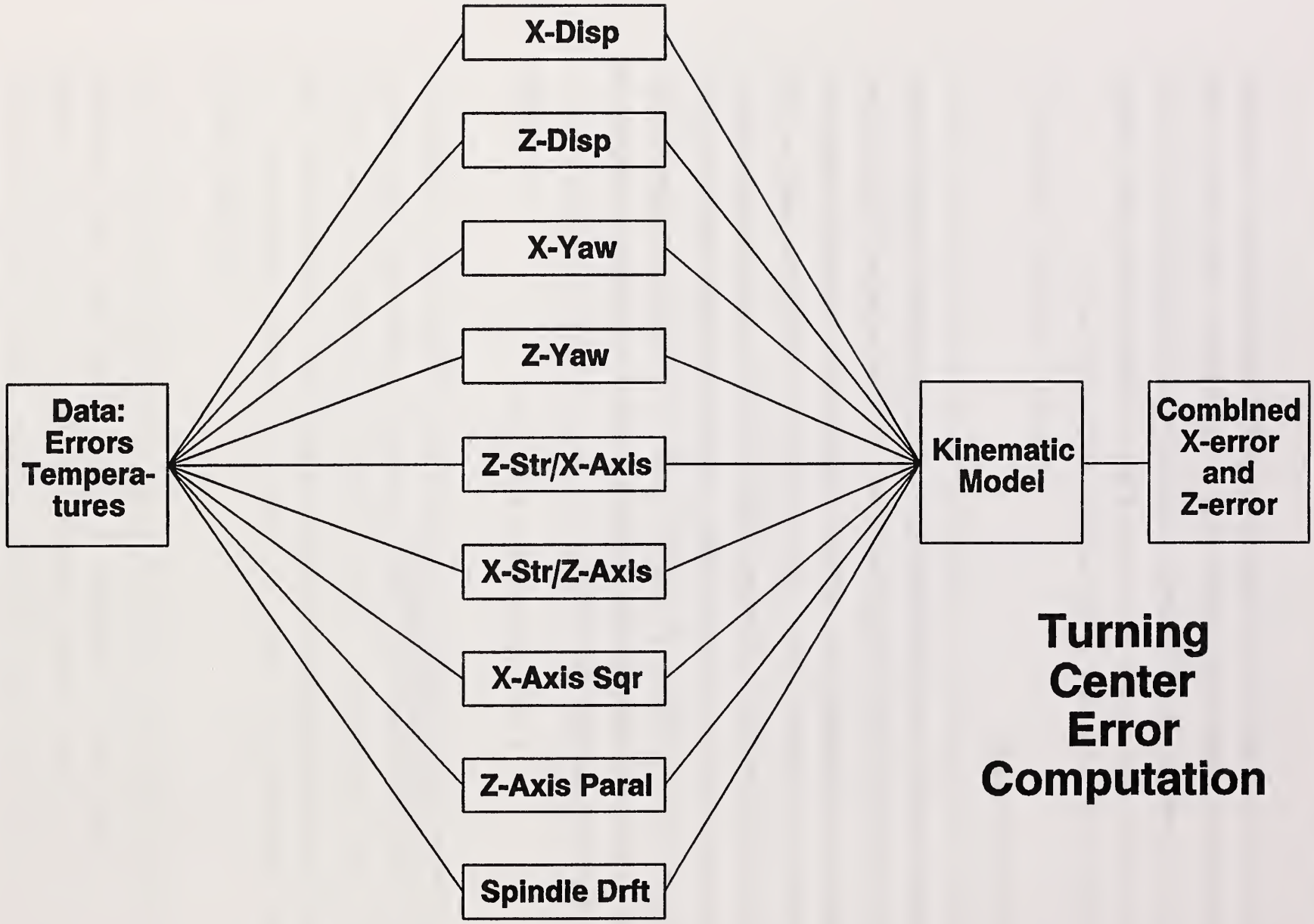

Figure 1: The component errors that contribute to the combined X and Z errors.

3.3.2 Angular Errors

Angular errors are rotational errors caused by geometric inaccuracies of the slideways and the misalignment in the assemblies of structural elements of the machine tool. The three rotational errors around the three orthogonal axes of the machine slide are defined as yaw, pitch and roll. Yaw error is the rotational error of the slide around the axis perpendicular to the plane in which the axis of motion lies. Pitch error is the rotational error of the slide around the axis which is in the plane of motion and perpendicular to the axis of motion of the slide. Roll error is the rotational error of the slide around the axis of motion. The contributions of all three types of rotational errors to the resultant error are significant in three or more axis machining centers. However, in turning centers, the contributions of roll and pitch errors to the resultant positioning error of the cutting tool are in the nonsensitive direction, thus creating second order errors. The nonsensitive direction in turning centers is the direction perpendicular to the plane in which the two machine slides are moving. Therefore, yaw error measurements are sufficient for resultant error calculations for turning centers of the type considered here. The yaw errors for an NC lathe have been found to be of the form:

$$\epsilon_y = a_0 + a_1 x + a_2 x^2 + b_1 T \tag{9}$$

where the subscript y represents the yaw error.

3.3.3 Straightness-Parallelism-Orthogonality

Slide straightness error is the nonlinear translational movement of the slide in the two orthogonal directions other than its axis of motion. Although straightness errors are translation errors rather than rotational their functional form for an NC lathe have taken a similar one to the form of angular errors described in section 3.3.2. Parallelism and orthogonality describe the angular orientation of the machine axes with respect to each other. They have been found to be polynomials with temperature dependence alone (see Donmez [5]).

3.3.4 Spindle Thermal Drift

Spindle thermal drift is the axial and radial translations along the axes of the machine tool along with tilt motions in two orthogonal planes. These are thermally dependent functions only.

4.0 Some Regression Approach Shortcomings

For the turning center error functions described above two data sets were developed for each error term (Gilsinn, et.al. [11]) These two data sets represented both forward and backward motions along the axis of motion. The equation fitting process was one of trial and error. Engineering judgement helped determine the appropriate variables entering each regression equation. This judgement is based upon the knowledge of cause and effect relationships between temperature rise and expansion of metal. The procedure did not lend itself to be automated, however.

Without engineering judgement, the regression process can become very unwieldy. In particular for the turning center studied 40 thermocouples were used. Approximately 600 data samples were taken along each axis of motion. A data sample included a nominal position along an axis of motion and readings from the 40 thermocouples along with the measured error. Thus the initial table for regression would be approximately 600 rows by 42 columns.

If a straightforward regression is required, to find equations up to quadratic order would require adding 42 x 42 more columns to the data matrix to account for all possible product terms. The end result would be a matrix of approximately 600 rows by 1800 columns. Standard regression packages would require the inversion of an 1800 x 1800 matrix to determine the regression coefficients. This is not a reasonable approach.

A stepwise regression procedure could begin by examining the correlation matrix between all of the 42 variables and select the first variable to enter the regression by the highest correlation coefficient. Next, regression would be plotted against each of the other variables to determine whether any of the other variables would likely enter the regression. If this process indicated another variable could be added to the regression equation then another regression fit would be performed with two variables. This process would continue until all variables are used up. Next columns for all of the products would be added to the process. Again the data matrix would expand to approximately 1800 columns.

It is clear that standard regression procedures are cumbersome when there are a large number of variables involved. Physical intuition helps reduce the effort but cannot be easily introduced into

an automated means of generating the error component function maps. For this reason neural networks are a possible alternative for generating component error maps.

5.0 Neural Networks

Neural Networks are linkages or networks of artificial processing units (or neurons) that model the behavior of biological neural systems. Early studies of biological nervous systems guided the structuring of neural networks. They can also be thought of as a weighted linking of distributed adaptable processing units. These networks are often "taught" by presenting examples of inputs and outputs to them and adjusting the weights and adapting the processing units so the neural network reproduce as nearly as possible the known outputs for a given set of inputs. The techniques, however, do not differ from those used in nonlinear regression.

Neural networks training can be thought of as solving a problem in classical approximation theory (see Girosi and Poggio [12]). This theory deals with the approximating or interpolating a continuous function f(x) by an approximating function F(w,x), where w is some vector of unknown parameters. The approximation problem is to find the vector w of parameters that makes F(w,x) match f(x) as closely as possible with respect to some predefined measure of closeness. Recent research (see G. Cybenko [13]) has linked the class of neural networks called multilayered, feedforward neural networks very closely with the classic approximation problem in the sense that any continuous function of several variables over finite intervals can be approximated by a linear combination of the functions defining the processing units. This will be discussed further below.

5.1 Parallel Distributed Processing Model

Neural networks can also be throught of from the point of view of parallel processing. According to Rumelhart, et al. [14] eight elements are relevant here:

1. Processing Units
2. A state of activation for each unit.
3. An output function for each unit.
4. A connectivity pattern between units

5. A rule for propagating patterns through the network.
6. An activation rule that combines inputs to a unit to produce an activation level for that unit.
7. A learning rule that modifies weights based on sample input and output data.
8. An environment in which the neural network operates.

5.1.1 Processing Units

Processing units perform the simple job of receiving input from their neighbors in the connectivity structure, compute an output and send it on to its neighbors. The system can be considered parallel in that each unit can carry out a computation at the same time.

Three types of units make up a network: input, hidden and output. Input units receive data from outside sources and output units send data to outside destinations. Hidden units are those whose inputs and outputs lie within the network itself.

5.1.2 State of Activation

As understood here, activation values are continuous values between 0 and 1. When a processing unit receives a weighted sum of inputs from connected neighboring cells it produces an output or activation level. This is a number between the minimum of 0 and the maximum of 1. Mathematically this can be expressed as:

$$y_i = f_i\left(\sum_{j=1}^{n} w_{ij}x_j - \theta_i\right) \tag{10}$$

where y_i represents the output activation signal from the i-th processing unit, w_{ij} represents the weight of the j to i interconnection, θ_i represents the threshold value of the i-th processing unit. The type of transformation given by

$$\sum_{j=1}^{n} w_{ij}x_j - \theta_i \tag{11}$$

is called an affine transformation.

The input to a processing unit can be either the output from connected neighboring units or directly from outside the network. The output from the i-th processing unit can be used either as input to neighboring units or as output from the network. The value of the weight w_{ij} determines how strongly the output of the j-th processing unit influences the activity of the i-th processing unit.

5.1.3 Output Function

The sigmoid (or S-shaped) output function is used in practice to determine the level of activation of a processing unit. For the i-th processing unit it is in general specified as:

$$y_i = \frac{1}{1 + e^{-ca_i}} \tag{12}$$

where

$$a_i = \sum_{j=1}^{n} w_{ij}x_j - \theta_i \tag{13}$$

The threshold θ_i acts as a filter for incoming signals and a_i is referred to as the activation of the i-th processing unit. It is transformed by the sigmoid function to determine the magnitude of the current output signal from the i-th unit. c is a constant which determines how steeply the i-th processing unit ascends from a minimum to a maximum value. It is taken as 1 in this report. The smaller it is the slower the ascent, the larger it is the more rapid the ascent. In particular for large values the sigmoid function begins to approximate a step function. The computational unit described in sections 5.1.1 to 5.1.3 is usually called a perceptron. This model is shown in Figure 2. This figure shows the weighted input to a processing unit from three sources. The output y is computed as a sigmoid function of an affine transformation of the inputs. The output is shown being sent to three other units.

5.1.4 Connectivity

Processing units are linked to each other. Units connected to others by links in the network are sometimes referred to as neighbors. Each link in the network has a weight assigned to it as

described above. The total pattern of links can be represented by specifying the weights for each of the connections in the network. A positive weight represents an excitation input and a negative weight represents an inhibitory input to a processing unit. The absolute value of the weight specifies the strength of the connection. The term fan-in is sometimes used to represent the number of units that generate inputs to a given unit and the term fan-out is the number of units affected by a given units output.

Network connectivity architectures can be divided into two types: recurrent and non-recurrent. Recurrent networks are those in which loops exist. That is, a units output can by way of various links in the network ultimately return to affect its own input. A network is non-recurrent if a units output does not affect its input. An important subclass of non-recurrent networks are those in which the processing units are organized in layers and only one way links are allowed between adjacent layers. These networks are called feedforward networks. Feedforward networks are called fully connected if each unit on a layer is connected to every unit on an adjacent layer. Thus if there are n units on one layer and m units on an adjacent layer then there are n x m connetions.

A feedforward network is in general a multilayered neural network in which perceptrons are arranged in layers, Where the output of one layer becomes the input to the next layer. Ordinarily the outputs of the final layer are weighted and summed together to produce the output, but the final layer can also act as perceptrons. This is the type of network used in the current study and is illustrated in Figure 3 in which there are three input nodes, four hidden nodes and two output nodes.

5.1.5 Rule of Propagation

A rule of propagation is one that specifies how output values from nodes are to be combined with the connectivities to produce inputs to neighboring nodes. Since the networks used in the current study are fully connected feedforward networks the overall propagation rule can be precisely specified.

The network used in this study and described here is a three layer neural net with one input layer, one hidden layer and one output layer. Let there be I input nodes, H hidden nodes and J output nodes. Each input node is connected to each of the hidden nodes and each of the hidden nodes

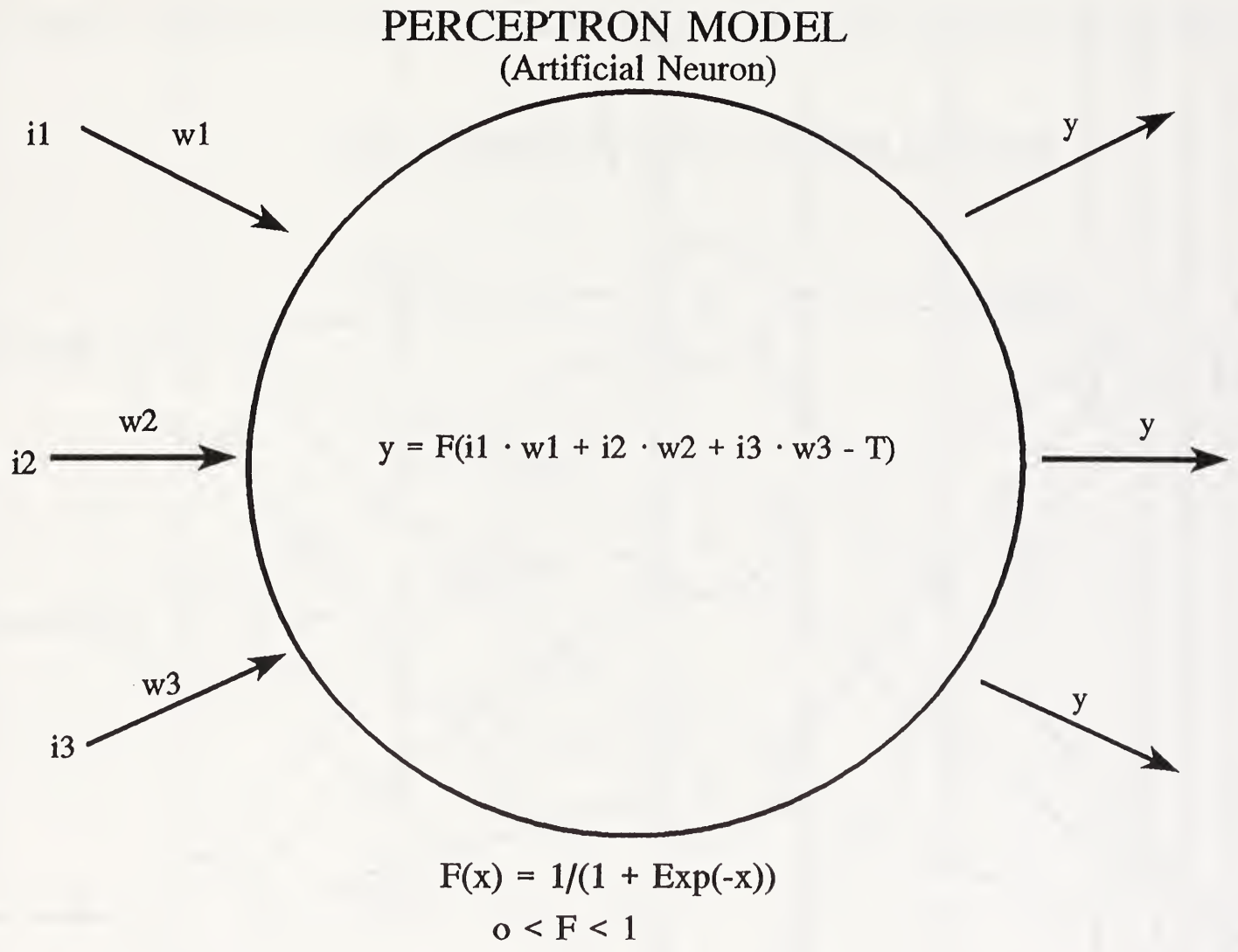

Figure 2. Perceptron model of a neural network processing unit.

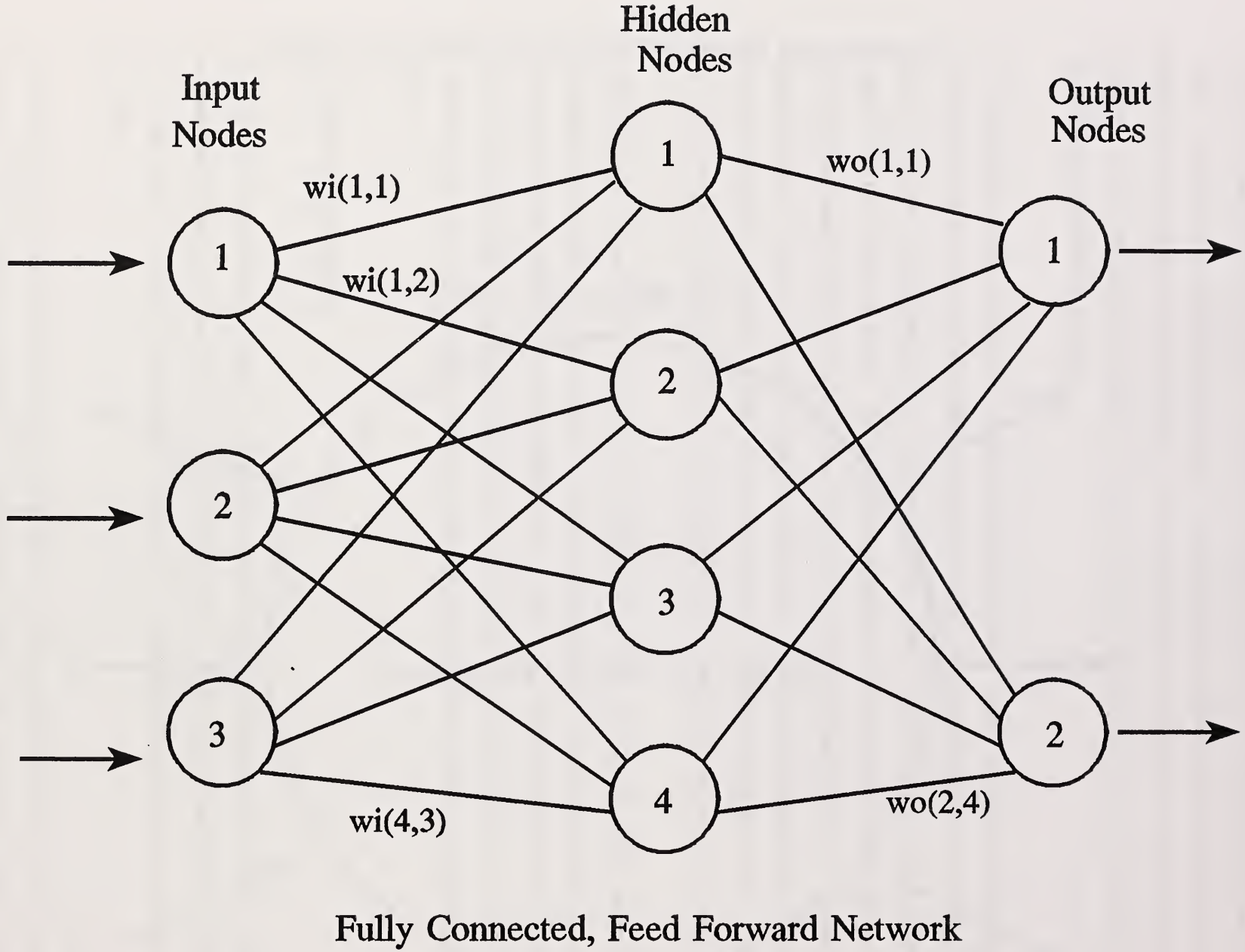

Figure 3. A sample of a fully connected feedforward network with three input nodes, four hidden nodes, and two output nodes.

is connected to every output node. This means that there are I x H x J connections between input nodes and output nodes. Let x_i be the external input to the network at the i-th input node where i = 1, ..., I. Let y_h be the output from h-th hidden node and z_j be the output from the j-th output node. Let $w1_{hi}$ be the weight on the link that connects the i-th input node with the h-th hidden node and let $w2_{jh}$ be the weight on the link that connects the h-th hidden node with the j-th output node. In order to simplify the notation slightly the threshold values in the nodal activations can be considered as weights in the sum and a dummy input node and hidden node can be added with a constant input value of 1. That is,

$$x_{I+1} = 1.0 \tag{14}$$

at the dummy I+1 input node and a constant output at the dummy hidden node of

$$y_{H+1} = 1.0 \tag{15}$$

The output function at each hidden node and output node will be the sigmoid function with $c = 1.0$.

With this notation the propagation rule can be stated as follows: For the j-th hidden node, j = 1, ..., J compute

$$y_h = f\left(\sum_{i=1}^{I+1} w1_{hi}x_i\right) \tag{16}$$

where $x_{I+1} = 1.0$. Next compute for each of the J output units the following:

$$z_j = f\left(\sum_{j=1}^{J+1} w2_{jh}y_h\right) \tag{17}$$

where $y_{H+1} = 1.0$. The output function at each processing node is given by the sigmoid function

$$f(s) = \frac{1}{1+e^{-s}} \tag{18}$$

Therefore given any input vector $\mathbf{x} = (x_1,...,x_I)$ the propagation rule above produces an output vector $\mathbf{z} = (z_1,...,z_J)$ at the final node layer.

5.1.6 Activation Rule

This is a rule by which the inputs to a processing nodes are combined and operated on by the nodal output function to produce a new activation level at the node. This process has been described mathematically in section 5.1.2.

5.1.7 Connectivity Modification

Changing the connectivity of a parallel distributed processing model changes its "knowledge". This can be done by:

1. Developing new connectivities.
2. Losing some connectivities.
3. Modifying the strengths of existing connectivities.

Developing new connectivities and losing connectivities can be considered special cases of 3 so that this study will consider only means of modifying existing strengths.

By far the most popular technique for modifying the connectivity strengths is called the Back-Propagation Algorithm. Other algorithms have recently shown better performance and one of these will be discussed below. These algorithms are essentially optimization algorithms.

5.1.8 The Usage Environment

Separate implementations of a feedforward neural net were applied to develop a selected number of component errors that enter into the kinematic turning center error correction model. Weights were developed to predict the linear displacement error of one axis of the machine using existing data. The inputs were axis locations and thermocouple readings. The outputs were individual error values for this particular error component.

If all of the error maps were developed by the method of neural nets, then the weights for each of the component errors entering the kinematic model would be looked up and the neural net propagation rule evaluated to produce error values for each component. These component errors

would be combined in the kinematic model to produce the spatial error of the turning center at the given nominal position and machine temperature state.

5.2 Learning Paradigm

The learning approach used to modify the connectivity strengths is based on the paradigm of associative learning. In this paradigm the neural net adapts its weights to produce a particular pattern of connectivities and activations on the output units when particular input data patterns are presented to the input units. The goal is to find a set of connectivity weightings and threshold values so that when a particular pattern of data appears at the input then the associated pattern of output data appears at the output units. The connectivity strengths are modified iteratively to try to teach the network (i.e. modify the connectivity strengths) so that during the training (or iterative modification) process the output data patterns for given training input patterns are made to more closely match the desired or target output data patterns for the given input patterns.

6.0 Gradient Descent Algorithms

Supervised training of a neural net involves the reduction of some error value. To assess the overall training (or weight adjustments) a cumulative error over all data patterns in the training set must be computed. In particular let there be P data patterns in the training set and J the number of output nodes. Let $\mathbf{z}_p = (z_{p1},...,z_{pJ})$ be the p-th output data pattern for the input vector $\mathbf{x}_p = (x_{p1},...,x_{pI})$. Furthermore, let $\mathbf{t}_p = (t_{p1},...,t_{pJ})$ be the desired or target data pattern for the p-th input $\mathbf{x}_p$. Then the root-mean-square normalized error for the training set is given by

$$E_{rms} = \frac{1}{\sqrt{PJ}} \sqrt{\sum_{p=1}^{P} \sum_{j=1}^{J} (t_{pj} - z_{pj})^2} \qquad (19)$$

For the purpose of developing the algorithms, however, let

$$E = \frac{1}{2}\sum_{p=1}^{P}\sum_{j=1}^{J}(t_{pj}-z_{pj})^2 \tag{20}$$

and then

$$E_{rms} = \frac{1}{\sqrt{PJ}}\sqrt{2E}. \tag{21}$$

In the next several sections a detailed discussion of the theoretical background of the computer code given in Appendix A will be given. Much of the description of the optimization algorithm is based on Møller [9]. The notation used in 6.2 is motivated by that used in Rumelhart [14]. The general optimization scheme is shown in Figure 4.

6.1 Algorithm Terminology

Vector notation will be used to simplify equations as much as possible. In particular, a network weight vector is a vector in real euclidean space R^N, where N is the total number of link weights and nodal threshold parameters. Let

$$w = (wl_{11},\ldots,wl_{1,I+1},wl_{21},\ldots,wl_{HI},\ldots,wl_{H,I+1},w2_{11},\ldots,w2_{1,H+1},\ldots)^T. \tag{22}$$

where wl_{hi} is the weight on the link that connects the i-th input unit with the h-th hidden unit. The superscript T refers to the transpose of the vector.

Let the cumulative error function E, equation (20) in section 6.0, be considered a function of w and denoted by $E(w)$. The gradient $E'(w)$, is given by

$$E'(w) = \left(\frac{\partial E}{\partial wl_{11}},\ldots,\frac{\partial E}{\partial wl_{1,I+1}},\frac{\partial E}{\partial wl_{21}},\ldots,\frac{\partial E}{\partial wl_{HI}},\ldots,\frac{\partial E}{\partial wl_{H,I+1}},\frac{\partial E}{\partial w2_{11}},\ldots,\frac{\partial E}{\partial w2_{1,H+1}},\ldots\right)^T \tag{23}$$

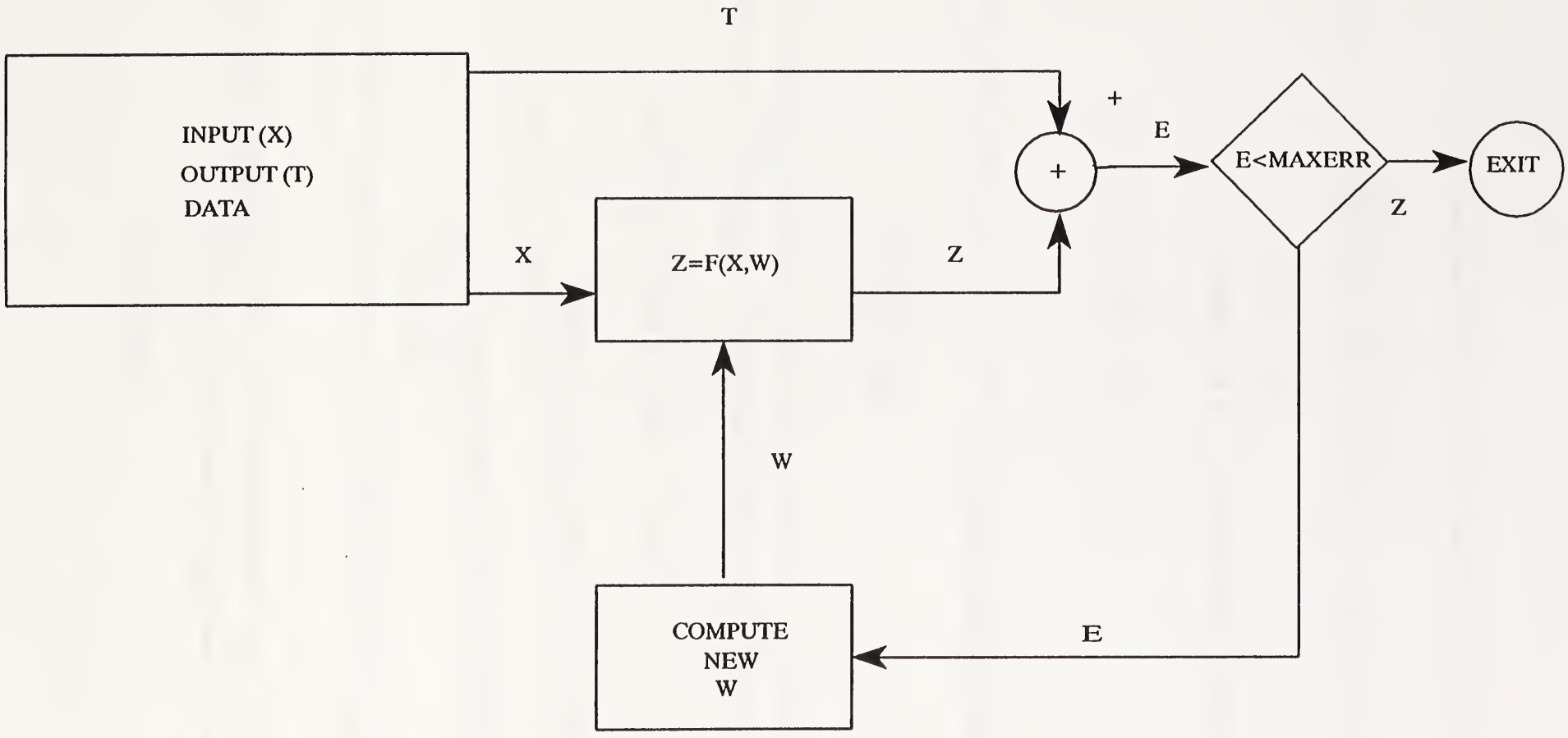

Figure 4. Schematic of the optimization strategy.

The sum and difference of two vectors, **w**, **v**, in R^N are given by

$$\begin{aligned} w + v &= (w_1 + v_1, \ldots, w_N + v_N)^T \\ w - v &= (w_1 - v_1, \ldots, w_N - v_N)^T \end{aligned} \tag{24}$$

The inner product of two vectors, **w**, **v**, in R^N is given by

$$w^T v = \sum_{i=1}^{N} w_i y_i \tag{25}$$

The length or norm of **w** is given by

$$|w| = \sqrt{\sum_{i=1}^{N} w_i^2} \tag{26}$$

The Taylor expansion in vector form for E(**w**) can be expressed as

$$E(w+y) = E(w) + E'(w)^T y + \frac{1}{2} y^T E''(w) y + \ldots \tag{27}$$

where E"(**w**) is an N x N matrix called the Hessian.

An N x N matrix A is said to be positive definite if

$$y^T A y > 0 \tag{28}$$

for all **y** in R^N.

Finally, let $\mathbf{p}_1, \ldots, \mathbf{p}_k$ be non-zero vectors in R^N. This set of vectors is said to be a conjugate system with respect to a non-singular symmetric N x N matrix A if

$$p_i^T A p_j = 0 \tag{29}$$

for all i,j = 1,...,k, $i \neq j$.

6.2 Computing the Gradient

To begin the computation of the gradient, E'(w), note that if f(u) is the sigmoid function

$$y = f(u) = \frac{1}{1+e^{-u}} \tag{30}$$

then

$$\frac{dy}{du} = f'(u) = f(u)(1-f(u)) = y(1-y) \tag{31}$$

For the fully connected feedforward network described in 5.1.5 introduce the notation

$$sl_h = \sum_{i=1}^{I+1} wl_{hi} x_{pi} \tag{32}$$

where h = 1,2,...,H for the hidden units and p represents the p-th pattern. The output of the h-th hidden unit is then given by

$$y_{ph} = f(sl_h) = \frac{1}{1+e^{-sl_h}} \tag{33}$$

The derivative of y_{ph} with respect to sl_h is then given by

$$\frac{dy_{ph}}{dsl_h} = y_{ph}(1-y_{ph}) \tag{34}$$

from the property of the sigmoid function derivative. This derivative is referred to as hderiv(h) in the program given in Appendix A.

For each of the output units, j = 1,2,...,J, let

$$s2_j = \sum_{h=1}^{H+1} w2_{jh} y_{ph} \tag{35}$$

The output of the sigmoid function is then given by

$$z_{pj} = f(s2_j) = \frac{1}{1+e^{-s2_j}} \tag{36}$$

The derivative is given by

$$\frac{dz_{pj}}{ds2_j} = z_{pj}(1-z_{pj}) \tag{37}$$

and is referred to as oderiv(j) in the program in Appendix A.

Let the contribution to the total error by the p-th data pattern be given by

$$E_p = \frac{1}{2}\sum_{j=1}^{J}(t_{pj}-z_{pj})^2 \tag{38}$$

The total error is then given by

$$E = \sum_{p=1}^{P} E_p \tag{39}$$

For notation let

$$\delta_{zj}^{p} = -\frac{\partial E_p}{\partial s2_j} \tag{40}$$

Note that the gradient component

$$\frac{\partial E}{\partial w2_{jh}} = \sum_{p=1}^{P}\frac{\partial E_p}{\partial w2_{jh}} \tag{41}$$

depends on only the sum $s2_j$ for the j-th output unit since the output at the j-th output unit depends only on $w2_{jh}$, h = 1,...,H. Therefore by the chain rule of partial differentiation

$$\frac{\partial E_p}{\partial w2_{jh}} = \frac{\partial E_p}{\partial s2_j}\frac{\partial s2_j}{\partial w2_{jh}} \tag{42}$$

But from the definition of $s2_j$,

$$\frac{\partial s2_j}{\partial w2_{jh}} = y_{ph} \tag{43}$$

Therefore

$$\frac{\partial E_p}{\partial w2_{JH}} = -\delta^p_{zj} y_{ph} \tag{44}$$

To complete this, one must compute δ^p_{zj}. Referring to the definition of E_p

$$\frac{\partial E_p}{\partial s2_j} = -(t_{pj} - z_{pj})\frac{dz_{pj}}{ds2_j} \tag{45}$$

since the output at the j-th output unit, z_{pj}, is the only one affected by $s2_j$. But then

$$\frac{\partial E_p}{\partial s2_j} = -(t_{pj} - z_{pj}) z_{pj}(1 - z_{pj}) \tag{46}$$

and therefore

$$\delta^p_{zj} = (t_{pj} - z_{pj}) z_{pj}(1 - z_{pj}) \tag{47}$$

This is referred to as delta2(p,j) in the program in Appendix A. Finally,

$$\frac{\partial E}{\partial w2_{jh}} = -\sum_{p=1}^{P} \delta^p_{zj} y_{ph} \tag{48}$$

In the program in Appendix A

$$gm2(j,h) = -\frac{\partial E}{\partial w2_{jh}} \tag{49}$$

To compute the other components of the gradient

$$\frac{\partial E}{\partial w1_{hi}} = \sum_{p=1}^{P} \frac{\partial E_p}{\partial w1_{hi}} \tag{50}$$

note that $w1_{hi}$ contributes only to the sum $s1_h$. Therefore, from the chain rule,

$$\frac{\partial E_p}{\partial w1_{hi}} = \frac{\partial E_p}{\partial s1_h} \frac{\partial s1_h}{\partial w1_{hi}} \tag{51}$$

But

$$\frac{\partial s1_h}{\partial w1_{hi}} = x_{pi} \tag{52}$$

Then, if one defines

$$\delta^p_{yh} = -\frac{\partial E_p}{\partial s1_h} \tag{53}$$

the equation above yields

$$\frac{\partial E_p}{\partial w1_{hi}} = -\delta^p_{yh} x_{pi} \tag{54}$$

and it is left to compute δ^p_{yh}. Since $s1_h$ only affects the output of the h-th hidden unit one can use the chain rule and compute

$$\frac{\partial E_p}{\partial s1_h} = \frac{\partial E_p}{\partial y_{ph}} \frac{\partial y_{ph}}{\partial s1_h} \tag{55}$$

But then

$$\frac{\partial E_p}{\partial s1_h} = \frac{\partial E_p}{\partial y_{ph}} y_{ph}(1 - y_{ph}) \tag{56}$$

From the definition of E_p

$$\frac{\partial E_p}{\partial y_{ph}} = -\sum_{j=1}^{J} (t_{pj} - z_{pj}) \frac{\partial z_{ph}}{\partial y_{ph}} \tag{57}$$

But

$$\frac{\partial z_{pj}}{\partial y_{ph}} = \frac{\partial z_{pj}}{\partial s2_j} \frac{\partial s2_j}{\partial y_{ph}} \tag{58}$$

or

$$\frac{\partial z_{pj}}{\partial y_{ph}} = z_{pj} (1 - z_{pj}) w2_{jh} \tag{59}$$

Then

$$\frac{\partial E_p}{\partial y_{ph}} = -\sum_{j=1}^{J} (t_{pj} - z_{pj}) z_{pj} (1 - z_{pj}) w2_{jh} \tag{60}$$

Then

$$\frac{\partial E_p}{\partial s1_h} = -y_{ph} (1 - y_{ph}) \sum_{j=1}^{J} (t_{pj} - z_{pj}) z_{pj} (1 - z_{pj}) w2_{jh} \tag{61}$$

or

$$\delta_{yh}^{p} = y_{ph} (1 - y_{ph}) \sum_{j=1}^{J} \delta_{zj}^{p} w2_{jh} \tag{62}$$

which is referred to as delta1(p,h) in the program and finally

$$\frac{\partial E}{\partial w1_{hi}} = -\sum_{p=1}^{P} \delta_{yh}^{p} x_{pi} \tag{63}$$

Note that in the program in Appendix A

$$gm1(h,i) = -\frac{\partial E}{\partial w1_{hi}} \tag{64}$$

The order in which the gradient components are compute is a reverse order. That is, for each pattern p, δ^p_{zj} is computed for each output unit, j = 1,...,J. Then the gradient component

$$\frac{\partial E}{\partial w2_{jh}} = -\sum_{p=1}^{P} \delta^p_{zj} y_{ph} \tag{65}$$

is computed. Next, for each hidden unit, h = 1,...,H, one computes

$$\delta^p_{yh} = y_{ph}(1 - y_{ph}) \sum_{j=1}^{J} \delta^p_{zj} w2_{jh} \tag{66}$$

and sets

$$\frac{\partial E}{\partial w1_{hi}} = -\sum_{p=1}^{P} \delta^p_{yh} x_{pi} \tag{67}$$

This is the reason that this method of computing the gradient is referred to as the method of back propagation. The term delta rule is sometimes used for obvious reasons. The implementation of this computation is given in the subroutine FORWARD in Appendix A.

6.3 Descent Strategy

The idea of the strategy used to minimize the error function E(**w**) is given in the following steps:

1. Select an initial weight vector $\mathbf{w}_1$ and set n = 1.

2. Determine a search direction $\mathbf{p}_n$ and a step size α_n so that if $E(\mathbf{w}_n + \Delta\mathbf{w}_n) < E(\mathbf{w}_n)$ where $\Delta\mathbf{w}_n = \alpha_n\mathbf{p}_n$.

3. Compute $E'(\mathbf{w}_n)$ and update the current weight vector: $\mathbf{w}_{n+1} = \mathbf{w}_n + \Delta\mathbf{w}_n$.

4. If $E'(w_n) \neq 0$ then update the iteration counter $n = n + 1$ and go back to step 2. Otherwise, return w_{n+1} as the desired weight vector that minimizes E(w).

Various implementations of this strategy differ at step 2 in how to select a search direction and step size.

6.4 Back Propagation Algorithm

The back propagation terminology should be applied strictly to the method of computing the gradient of the error function as done in section 6.2. It is, though, generally applied to the method of updating the weight vector by stepping in the dirction of the steepest descent.

To identify each iterative step, n will be used to designate the n-th step. At the n-th step the weight update rule for the back propagation algorithm is

$$\Delta w1_{hi}(n) = -\eta \frac{\partial E}{\partial w1_{hi}}(n) = \eta \sum_{p=1}^{P} \delta_{yh}^{p}(n) x_{pi} \tag{68}$$

and

$$\Delta w2_{jh}(n) = -\eta \frac{\partial E}{\partial w2_{jh}}(n) = \eta \sum_{p=1}^{P} \delta_{zj}^{p}(n) x_{ph} \tag{69}$$

where η is the multiplier that specifies the step size. It is sometimes referred to as the learning rate. The index n indicates values computed at the n-th stage of the algorithm. The updated weights are often computed as

$$\begin{aligned} \Delta w1_{hi}(n) &= -\eta \frac{\partial E}{\partial w1_{hi}}(n) + \alpha \Delta w1_{hi}(n-1) \\ \Delta w2_{jh}(n) &= -\eta \frac{\partial E}{\partial w2_{jh}}(n) + \alpha \Delta w2_{jh}(n-1) \end{aligned} \tag{70}$$

The last term in the sum is called the momentum term and is used as a means of increasing the convergence rate. The back propagation update procedure is equivalent to setting

$$p_k = -E'(w) \tag{71}$$

in 6.3 and $\alpha_k = \eta$, a constant. This does not include the momentum term. The end result is a minimization procedure based on the first order Taylor series approximation

$$E(w+y) \approx E(w) + E'(w)^T y \tag{72}$$

This linear approximation is one reason for the poor convergence record of the back propagation algorithm. The use of a constant step length does not give the algorithm the robustness of an adaptively adjusted step size. Adding the momentum term as an attempt to make the algorithm approximately second order adds another constant multipier that again adds to the lack of robustness of the algorithm. In fact if the learning rate parameter is not selected properly oscillations in the error function can be generated without producing any descent.

6.5 Scaled Conjugate Gradient Algorithm

The algorithm in this section is due to Møller [9].

Assume that the error function E from equation (20) can be approximated locally (i.e. in a neighborhood of a weight vector **w**) by a quadratic function of the form

$$E_{qw}(y) = E(w) + E'(w)^T y + \frac{1}{2} y^T E''(w) y \tag{73}$$

where the subscript q stands for quadratic. The minimum of $E_{qw}(\mathbf{y})$ is among the *critical* points which are found by solving

$$E_{qw}^{(1)}(y) = E''(w)y + E'(w) = 0 \tag{74}$$

for **y**. If the Hessian, E"(**w**), is positive definite then there is a unique global minimum. The minimum can be computed iteratively. Two issues arise: How to select the search directions and how to maintain the local Hessian positive definite for a non-quadratic function.

Let $\mathbf{p}_1,...,\mathbf{p}_N$ be a conjugate system with respect to the Hessian. $\mathbf{p}_1,...,\mathbf{p}_N$ forms a basis for R^N. The vector from the initial point $\mathbf{y}_1$ to the minimum $\mathbf{y}_*$ can be expressed as a unique linear combination

of $\mathbf{p}_1,\ldots,\mathbf{p}_N$,

$$y_* - y_1 = \sum_{i=1}^{N} \alpha_i p_i \tag{75}$$

for some α_i's real. The α_i's can be computed as follows: Multiply the Eq. (75) by $\mathbf{p}^T_j E''(\mathbf{w})$ and get

$$p_j^T (E''(w) y_* - E''(w) y_1) = \alpha_j p_j^T E''(w) p_j \tag{76}$$

Using

$$E'(w) = -E''(w) y_* \tag{77}$$

gives

$$p_j^T (-E'(w) - E''(w) y_1) = \alpha_j p_j^T E''(w) p_j \tag{78}$$

Noting that

$$E^{(1)}_{qw}(y_1) = E''(w) y_1 + E'(w) \tag{79}$$

one can solve for α_j as

$$\alpha_j = \frac{-p_j^T E^{(1)}_{qw}(y_1)}{p_j^T E''(w) p_j} \tag{80}$$

Therefore the minimum of the quadratic, $\mathbf{y}_*$, can be computed in N iterative steps. This can be restated as follows. Let $\mathbf{p}_1,\ldots,\mathbf{p}_N$ be a conjugate system relative to the Hessian and let $\mathbf{y}_1$ be an initial point in the weight space. Determine the points $\mathbf{y}_2,\ldots,\mathbf{y}_{N+1}$ recursively by

$$y_{k+1} = y_k + \alpha_k p_k \tag{81}$$

where

$$\alpha_k = \frac{\mu_k}{\delta_k} \tag{82}$$

and

$$\mu_k = -p_k^T E_{qw}^{(1)}(y_k)$$
$$\delta_k = p_k^T E''(w) p_k \tag{83}$$

Then y_{N+1} minimizes $E_{qw}(y)$.

It is not necessary to assume that a conjugate system $p_1,...,p_N$ be known ahead of time. It is possible to compute them iteratively along with the computation of $y_2,...,y_{N+1}$ (Hestenes[15]). In fact, begin with y_1 in the weight space and take initially

$$p_1 = r_1 = -E_{qw}^{(1)}(y_1) \tag{84}$$

the initial steepest descent. Then compute recursively

$$y_{k+1} = y_k + \alpha_k p_k \tag{85}$$

where α_k is given above in Eq. (82) and

$$r_{k+1} = -E_{qw}^{(1)}(y_{k+1}) \tag{86}$$

Set

$$p_{k+1} = r_{k+1} + \beta_k p_k \tag{87}$$

where

$$\beta_k = \frac{|r_{k+1}|^2 - r_{k+1}^T r_k}{p_k^T r_k} \tag{88}$$

That is, p_{k+1} is determined recursively as a linear combination of the current descent direction, r_{k+1}, and the previous direction p_k.

Since the error function, E(w), is not necessarily quadratic the algorithm will not necessarily converge in N steps. If it does not, then the algorithm is started again with the steepest descent at the current point and continued.

At this point if one assumes that E"(w) is positive definite then the descent strategy from section 6.3 for the conjugate gradient algorithm can be stated as follows:

1. Initialize. Set the iteration counter k = 1. Select an initial weight vector $\mathbf{w}_1$. Select the steepest descent at $\mathbf{w}_1$ as the initial direction

$$p_1 = r_1 = -E'(w_1) \tag{89}$$

2. Calculate the current step length

$$\begin{aligned} s_k &= E''(w_k)p_k \\ \delta_k &= p_k^T s_k \\ \mu_k &= p_k^T r_k \\ \alpha_k &= \frac{\mu_k}{\delta_k} \end{aligned} \tag{90}$$

3. Update the weight vector

$$w_{k+1} = w_k + \alpha_k p_k \tag{91}$$

4. Get the new descent direction

$$r_{k+1} = -E'(w_{k+1}) \tag{92}$$

5. If the iteration counter k is a multiple of N then restart the algorithm by setting the direction to the current steepest descent and continue:

$$p_{k+1} = r_{k+1} \tag{93}$$

6. If k is not a multiple of N then compute a new conjugate direction by setting

7. If the steepest descent $\mathbf{r}_k \neq 0$ then increment the iteration counter k = k + 1 and go back to step 2 to get a new step length.

$$\beta_k = \frac{|r_{k+1}|^2 - r_{k+1}^T r_k}{\mu_k}$$
$$p_{k+1} = r_{k+1} + \beta_k p_k \tag{94}$$

8. If $r_k = 0$ then return w_{k+1} as the minimum.

For each iteration of this version of the algorithm the Hessian $E''(w_k)$ must be computed and stored. This can be a large matrix. Another approach is to estimate s_k in step 2 above by the difference quotient (Hestenes [15])

$$s_k = E''(w_k)p_k \approx \frac{E'(w_k + \sigma_k p_k) - E'(w_k)}{\sigma_k} \tag{95}$$

where $0 < \sigma_k << 1$. But this does not totally solve the minimization problem.

The previous algorithm assumed that $E''(w)$ was globally positive definite and that the quadratic approximation is a good approximation of $E(w)$. Two devices have to be introduced to assure this. The first device is to introduce a positive scalar parameter, λ_k, by setting

$$s_k = \frac{E'(w_k + \sigma_k p_k) - E'(w_k)}{\sigma_k} + \lambda_k p_k \tag{96}$$

The parameter λ_k (called a Marquardt-Levenberg parameter) is chosen to control the positive definiteness of $E''(w_k)$. The second device is introduced to control how good the quadratic approximation is. This is done by setting

$$\Delta_k = \frac{E(w_k) - E(w_k + \alpha_k p_k)}{E(w_k) - E_{qw}(\alpha_k p_k)} \tag{97}$$

Δ_k measures how well $E_{qw}(\alpha_k p_k)$ approximates $E(w_k + \alpha_k p_k)$. If Δ_k is close to 1 it indicates a good approximation.

At each iteration the sign of δ_k in step 2 above tells the definiteness of $E''(w_k)$. If $\delta_k > 0$ it is positive definite. If $\delta_k \leq 0$ then λ_k must be adjusted to make $\delta_k > 0$. The adjusted value is computed as

$$\overline{\lambda_k} = 2\left(\lambda_k - \frac{\delta_k}{|p_k|^2}\right) \tag{98}$$

A new δ_k is computed as

$$\overline{\delta_k} = \delta_k + (\overline{\lambda_k} - \lambda_k)|p_k|^2 = -\delta_k + \lambda_k|p_k|^2 \tag{99}$$

The new step size is then given by

$$\alpha_k = \frac{\mu_k}{\delta_k} = \frac{\mu_k}{p_k^T s_k + \lambda_k|p_k|^2} \tag{100}$$

As a note here, the bigger λ_k the smaller the step size α_k. The strategy (see Fletcher [16]) in adjusting λ_k also involves testing the measure Δ_k. In particular, if $\Delta_k > 0.75$, λ_k can be relaxed by setting

$$\lambda_k = \frac{1}{2}\lambda_k \tag{101}$$

This allows longer step lengths. If $\Delta_k < 0.25$ then λ_k is made larger by

$$\lambda_k = 4\lambda_k \tag{102}$$

This forces shorter steps in regions where the quadratic assumption is weaker.

The final scaled conjugate gradient algorithm (Møller [9]) can now be stated:

1. a. Select an initial weight vector $\mathbf{w}_1$ and scalars $\sigma > 0$, $\lambda_1 > 0$, and $\overline{\lambda}_1 = 0$.
   b. Set the initial search direction to the steepest descent and the iteration counter, k, to 1, i.e. $\mathbf{p}_1 = \mathbf{r}_1 = -E'(\mathbf{w}_1)$ and $k = 1$.
   c. Set a logical flag to true to indicate that a successful step to reduce the error function can be made: success = .true.

2. If a successful reduction in error cannot be made, i.e. success = .false., then go on to step 3. If a successful reduction in error can be made, i.e. success = .true., calculate new second order terms:

$$\begin{aligned} \sigma_k &= \frac{\sigma}{|p_k|} \\ s_k &= \frac{E'(w_k + \sigma_k p_k) - E'(w_k)}{\sigma_k} \\ \delta_k &= p_k^T s_k \end{aligned} \tag{103}$$

3. Scale the approximate Hessian, s_k, and the definiteness parameter, δ_k:

$$\begin{aligned} s_k &= s_k + (\lambda_k - \overline{\lambda_k}) p_k \\ \delta_k &= \delta_k + (\lambda_k - \overline{\lambda_k}) |p_k|2 \end{aligned} \tag{104}$$

4. If the approximate Hessian is positive definite, $\delta_k > 0$, then go to step 5 to calculate a new step size. If the approximate Hessian is not positive definite, $\delta_k \leq 0$, then make it positive definite:

$$\begin{aligned} s_k &= s_k + \left(\lambda_k - 2\frac{\delta_k}{|p_k|}2\right) p_k \\ \overline{\lambda_k} &= 2\left(\lambda_k - \frac{\delta_k}{|p_k|}2\right) \\ \delta_k &= -\delta_k + \lambda_k |p_k|2 \\ \lambda_k &= \overline{\lambda_k} \end{aligned} \tag{105}$$

5. Calculate the step size:

$$\begin{aligned} \mu_k &= p_k^T r_k \\ \alpha_k &= \frac{\mu_k}{\delta_k} \end{aligned} \tag{106}$$

6. Calculate the comparison parameter that measures how close the error function is to quadratic at the current weight vector:

$$\Delta_k = \frac{2\delta_k[E(w_k) - E(w_k + \alpha_k p_k)]}{\mu_k^2} \tag{107}$$

7. If $\Delta_k \geq 0$ then a successful reduction in error can be made. Set success = .true. and

$$\begin{aligned} w_{k+1} &= w_k + \alpha_k p_k \\ r_{k+1} &= -E'(w_{k+1}) \\ \overline{\lambda_k} &= 0 \end{aligned} \tag{108}$$

7a. Test whether k is a multiple of N.

7a.1. If so then restart the algorithm by setting the current direction to the current steepest descent, $p_{k+1} = r_{k+1}$ and go on to step 7b.

7a.2. If not then create a new conjugate direction to step in by setting:

$$\begin{aligned} \beta_k &= \frac{|r_{k+1}|2 - r_{k+1}^T r_k}{\mu_k} \\ p_{k+1} &= r_{k+1} + \beta_k p_k \end{aligned} \tag{109}$$

7b. If $\Delta_k \geq 0.75$ then the quadratic approximation is considered trustworthy and the scale parameter for the Hessian can be reduced by setting

$$\lambda_k = \frac{1}{2}\lambda_k \tag{110}$$

7c. If $\Delta_k < 0$ then a reduction in the error function is not possible. Set success = .false. and

$$\overline{\lambda_k} = \lambda_k \tag{111}$$

If there are too many failures in a row terminate the algorithm, otherwise continue to step 8.

8. If $\Delta_k < 0.25$ then the quadratic approximation is considered untrustworthy and the Hessian scale

parameter is increased to reduce the step size by setting

$$\lambda_k = 4\lambda_k \tag{112}$$

9a. If the steepest descent is nonzero, $r_k \neq 0$, then update the iteration counter, k = k+1, and go back to step 2.

9b. Otherwise, terminate the algorithm and return $\mathbf{w}_{k+1}$ as the weight vector for the desired minimum.

The implementation by Dr. James Blue at NIST of Møller's algorithm is given in the subroutine OPTWTS in Appendix A. For a comparison of the numerical advantages of the scaled conjugate gradient technique versus the straightforward back propagation method see Grother and Blue [8].

7.0 Adjusting for Ill Conditioning

The discussion in this section is based on the results of Tychonov and Arsenin [17]. Since some problems have solutions that are sensitive to small computational errors they have introduced a technique that can be used to reduce this sensitivity. It is called the method of regularization. There are other methods but in this section only the technique used in the program in Appendix A will be discussed. For a discussion of other methods see Saarinen, Bramley and Cybenko [18].

Let **w** be an n-dimensional real vector and **z** an m-dimensional real vector where n is greater than or equal to m. The object is to find **w** such that

$$F(w) = z \tag{113}$$

for a given **z**. The solution of this problem comes about by finding a map, R, such that

$$w = R(z). \tag{114}$$

Since there might be no unique **w** to solve (113), R may not be uniquely defined.

In general, R in equation (114) will be called stable if **w** depends continuously on **z**. More formally, define a metric in n-dimensional space by

$$\|w_1 - w_2\| = \sqrt{\sum_{i=1}^{n} (w_{1i} - w_{2i})^2} \tag{115}$$

For the moment assume that to every **z** there is a unique **w** such that **w** = R(**z**). The problem of finding R is said to be stable if, for every $\varepsilon > 0$, there exists a $\delta(\varepsilon)$ such that

$$\|z_1 - z_2\| \leq \delta(\epsilon) \Rightarrow \|w_1 - w_2\| \leq \epsilon \tag{116}$$

where $\mathbf{w}_1 = R(\mathbf{z}_1)$, $\mathbf{w}_2 = R(\mathbf{z}_2)$. Thus, small perturbations in **z** lead to small perturbations in **w**. A stable problem is also called a well posed problem.

The problem of finding an approximate solution of (113), given **z**, in an ill posed case, i.e. not stable as defined above, may be ambiguous in that there may be multiple solutions. Suppose that in (113) **z** is only known approximately as $\mathbf{z}_a$ where

$$\|z - z_a\| \leq \delta. \tag{117}$$

Let

$$Q_\delta = \{ w \mid \|F(w) - z_a\| \leq \delta \}. \tag{118}$$

The problem of finding an approximate solution for (113) reduces to selecting an appropriate **w** from Q_δ. This set may be too large so that not every element can be selected.

Suppose that $F(\mathbf{w}_0) = \mathbf{z}_0$. An operator $R(\mathbf{z},\alpha)$, depending on a parameter α, is called a regularizing operator for $F(\mathbf{w}) = \mathbf{z}$ if

(1) there is a $\delta_0 > 0$ such that the operator is defined for every $\alpha > 0$ and

$$\|z - z_0\| \leq \delta \leq \delta_0, \tag{119}$$

and

(2) there exists a function $\alpha = \alpha(\delta)$ such that for every $\varepsilon > 0$ there exists $\delta(\varepsilon) < \delta_0$ such that where $\mathbf{w}_\delta = R(\mathbf{z}_\delta,\alpha(\delta))$. This says that, given a regularizing operator for (113), if

$$\|z_\delta - z_0\| \le \delta(\epsilon) \Rightarrow \|w_\delta - w_0\| \le \epsilon, \tag{120}$$

$$\|z_\delta - z_0\| \le \delta \tag{121}$$

then $w_\delta = R(z_\delta,\alpha(\delta))$ can be taken as an approximate solution. This solution is called the regularized solution and α the regularizing parameter. The definition also implies that the existence of a regularizing operator defines a stable method of approximating a solution to (113). Therefore the problem of finding an approximate solution of (113) that is stable under small changes in the right hand side reduces to finding an appropriate regularization operator and determining the regularization parameter. This is called the Tychonov regularization method.

Define the function

$$\Omega(w) = \|w\|^2 \tag{122}$$

This is called a stabilizing function. The set of w such that

$$\|w\|^2 \le d \tag{123}$$

for a fixed d is closed and bounded. Equation (120) is taken as the stabilizing function and will be used in the following manner:

Assume $F(w) = z_0$ has a unique solution for the moment, called w_0. Let z_δ be an approximation to z_0 such that

$$\|z_\delta - z_0\| \le \delta. \tag{124}$$

Define

$$Q_\delta = \{ w \mid \|F(w) - z_\delta\| \le \delta \}. \tag{125}$$

The strategy for defining a regularization operator will be to look for elements in Q_δ that minimize $\Omega(w)$. If w_δ is such a number then the mapping $w_\delta = R(z_\delta,\delta)$ will be defined. It can be shown (see

Tychonov and Arsenin [17]) that $R(\mathbf{z},\delta)$ is a regularizing operator and can be taken as an approximate solution of (113) for $\mathbf{z} = \mathbf{z}_0$.

From this it is clear that the problem of finding an approximate solution of (113) with approximate right hand side $\mathbf{z}_\delta$ consists of finding $\mathbf{w}_\delta$ where

$$\Omega(w_\delta) = \min_{w \in Q_\delta} \Omega(w) \tag{126}$$

and

$$Q_\delta = \{ w \mid \| F(w) - z_\delta \| \leq \delta \}. \tag{127}$$

If $\mathbf{w} = 0$ is not an element of Q_δ then it can be shown (Tychonov and Arsenin [17]) that the greatest lower bound of $\Omega(w)$ on Q_δ is attained at an element w_δ such that

$$\| F(w_\delta) - z_\delta \| = \delta. \tag{128}$$

This says that the problem of solving (113) with approximate right hand side $\mathbf{z}_\delta$ is equivalent to the conditional extremum problem of finding

$$\min \| w \|^2 \tag{129}$$

subject to

$$\| F(w) - z_\delta \| = \delta. \tag{130}$$

The method of Lagrange multipliers is used to solve this problem by minimizing

$$L(w,\alpha) = \| F(w) - z_\delta \|^2 + \alpha \| w \|^2 \tag{131}$$

where α is determined by the condition that

$$\| F(w_\alpha) - z_\delta \| = \delta \tag{132}$$

and w_α is an element for which $L(w,\alpha)$ attains its greatest lower bound. Computationally α can be found by taking a sequence of α's

$$\alpha_k = \alpha_0 q^k \tag{133}$$

where α_0 is fixed and $0 < q < 1$ for $k = 0, 1, 2, \ldots$. For each α_k minimize (129) and calculate the differences

$$\| F(w_{\alpha_k}) - z_\delta \|. \tag{134}$$

Select α_k such that

$$\| F(w_{\alpha_k}) - z_\delta \| = \delta. \tag{135}$$

In practice though an α is selected experimentally (see Grother and Blue [8]).

The method of Lagrange multipiers is used in the program in Appendix A. In the subroutine FUNC the Lagrange multiplier is called wfactor and is read in from the run input file.

8.0 Statistical Goodness-of-Fit Measures

Since the neural net model assumed in the feedforward network is nonlinear the usual statistical goodness-of-fit tests applied in linear regression analysis are not directly applicable. However, the residuals between the target training data and the predicted data can be compared against a normal distribution. This comparison can be used to test whether the distribution of residuals forms a normal distribution with mean 0.

Before a test against a normal distribution is made the statistics to look at are: 1. Mean, 2. Standard deviation and 3. Minimum and Maximum values. The mean and standard deviation can be used to transform the residual errors given by

$$e(i) = \text{Target}(i) - \text{Pred}(i) \tag{136}$$

for i = 1 to the number of samples to a standardized form. If the residuals e(i) satisfy a normal distribution with sample mean $\bar{e}$ and sample standard deviation s, for sufficiently large samples,

$$E(i) = (e(i) - \bar{e})/s \tag{137}$$

approximates a standard normal distribution, i.e. a normal distribution with mean 0 and standard deviation 1. To test the hypothesis of normality the distribution E(i) can be compared to a standard normal distribution. The test can be performed by using the Kolmogorov-Smirnov test.

The Kolmogorov-Smirnov two-sample test is a statistical test of whether two independent samples have been drawn from the same population (or from populations with the same distribution). The two-tailed test is sensitive to any kind of difference in the distribution from which the two samples were drawn. The test is concerned with agreement between two cumulative distributions. If the two samples have in fact been drawn from the same population distribution then the cumulative distributions of both samples may be expected to be fairly close to each other. A large enough deviation between two sample cumulative distributions is evidence for rejecting the hypothesis that they are drawn from the same distribution.

The test involves specifying the cumulative frequency distribution which would occur under the theoretical distribution and comparing that with the observed cumulative distribution. The theoretical distribution represents what would be expected under the null hypothesis which is that the normalized residuals satisfy the normal distribution with mean 0 and standard deviation 1. The point at which these two distributions, theoretical and observed, show the greatest divergence is determined. This statistic is used to perform the test.

For comparing one data set E(x) to a known cumulative distribution the statistic of interest is

$$D = \max_{-\infty<x<\infty} | E(x) - P(x) | \tag{138}$$

The significance level of an observed value of D is given approximately by the equation

$$Q_{ks}(\lambda) = 2\sum_{j=1}^{\infty} (-1)^{j-1} e^{-2j^2\lambda^2} \tag{139}$$

The significance of an observed value of D is given approximately by

$$Prob(D \geq Observed\ D) = Q_{ks}(\sqrt{N}Observed\ D) \tag{140}$$

To compute the standardized cumulative normal one computes

$$F(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt \tag{141}$$

for $x \geq 0$ and $F(x) = 1 - F(-x)$ for $x < 0$. Instead of directly computing the standardized cumulative normal one Eq. (139), we can associate F(x) with the error function

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_{0}^{x} e^{-t^2} dt \tag{142}$$

by the relation $F(x) = 0.5(1+erf(x/\sqrt{2}))$ for $x \geq 0$ and $F(x) = 1 - F(-x)$ for $x < 0$. The error function can be approximated by a polynomial as given in the ERF subroutine of the accompanying program.

9.0 Geometric-Thermal Error Component Mapping

As shown earlier in Fig. 1, there are multiple sets of functions used as inputs to a kinematic model to predict work volume error. This study takes a limited subset of those data sets to demonstrate the nonlinear regression capabilities of a feedforward neural network to fit these data sets. Two data sets are used. The measurement given in these two data sets were taken at different times. They are the x-axis displacement errors moving vertically downward and vertically upward (see Figure 5). The data shows that the hysteresis errors in approaching a point on an axis must be mapped as two separate functions.

The criteria for a good fit are to have a distribution of the residuals after an optimization or test run that has a near zero mean and has a standard deviation of less than half a micrometer. The

standard deviation requirement was based on the resolution of the correction available for this machine tool.

9.1 X-Displacement (Down)

The data for this section was taken while the cross-slide of the lathe moved downwards.

The original data set included 595 records that listed nominal position, displacement error at that nominal position and 40 thermocouple readings. For the locations of the thermocouples see Figure 5 and Table 1. Every third point was selected for a testing set of data (198 points). The rest (397 points) were taken for training.

The total number of the records included 35 repetitions of 17 samples from -12.7 mm to -215.9 mm along the x-axis, with a sampling increment of -12.7 mm. The negative sign is used because the machine "home", or zero position, is approximately 386 mm above the spindle axis. A partial sample of the data set is shown in Table 2. The first row shows the number of data samples in the data set, the number of inputs (41, i.e. one nominal position and 40 thermocouples), the number of outputs (in this study 1), and a scale factor for unit conversion if necessary. The next row is a set of column titles. "Position" refers to the column beginning with -12.7 (nominal position in millimeters), "Diff" refers to the column beginning with -1.1631E-3 (the measured error in millimeters). 0,1,2 are headers for the next three columns. Note that the last one was wrapped around by the printer. These columns are the thermocouple readings in celsius for thermocouples 0, 1, 2. The columns for the other thermocouple readings are stacked in groups of 6 below the first 397 records. That is 397 records for thermocouples 3 through 8 are grouped after the first group, then 397 records for thermocouples 9 through 14, etc.

The first set of 397 records has the first column as an index of the order in the group of 17 nominal position samples. Thus 1 in column 1 indicates the first of 17 positions, 2 the second, etc. The group numbers repeat for every repetition of the 17 samples. Note that some of the indices are missing. These records have been extracted to form a file of test value to check the nonlinear

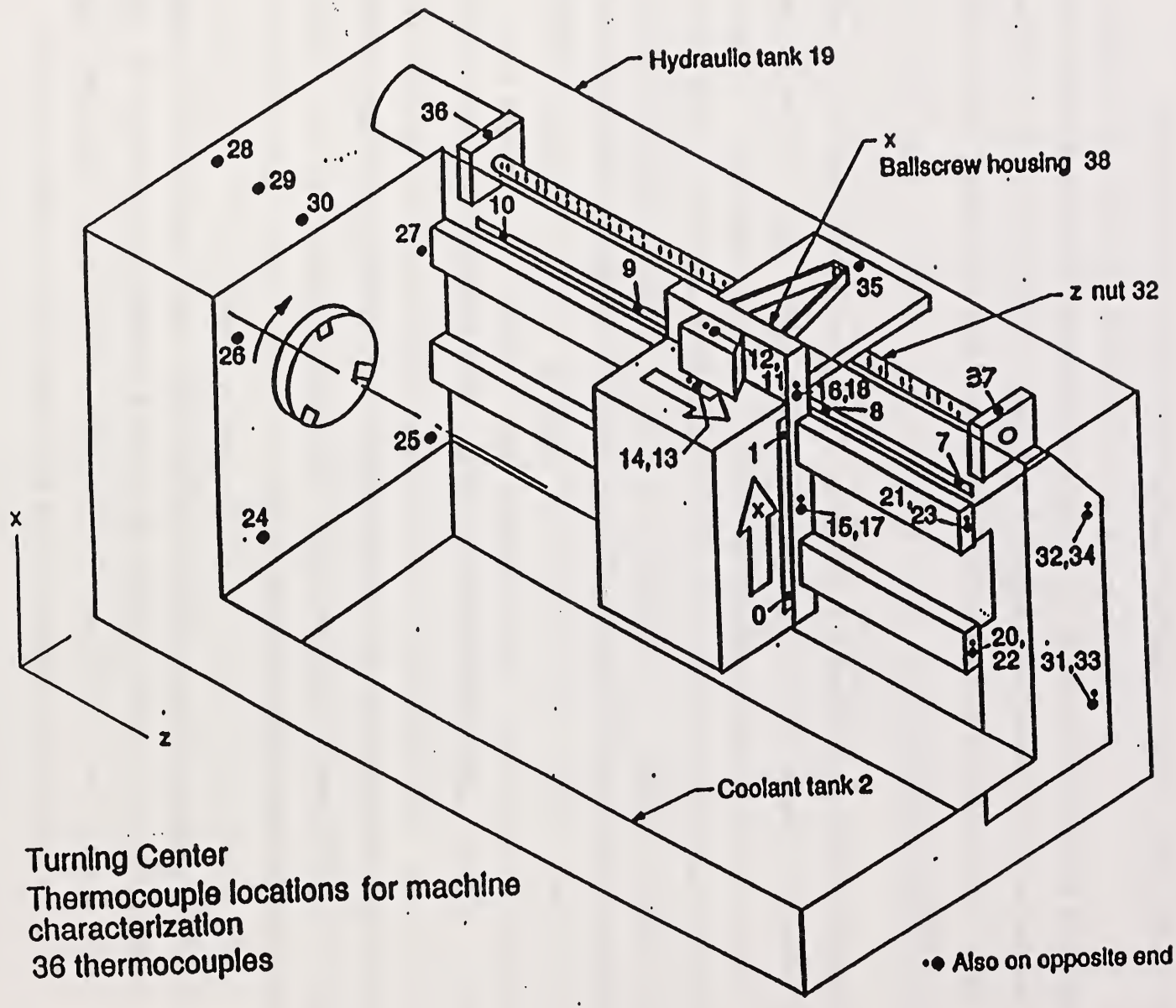

Figure 5. Turning Center with Thermocouple locations identified

NO.	LOCATION	NO.	LOCATION
0	Bottom of X-(Glass) Scale	20	Left End of Lower Z-Way
1	Top of X-Scale	21	Left End of Upper Z-Way
2	Coolant Tank	22	Right End of Lower Z-Way
3	Ambient	23	Right End of Upper Z-Way
4	Not Used	24	Lower FRont of Spindle Head
5	Not Used	25	Lower Rear of Spindle Head
6	Top Right of Bed	26	Upper Front of Spindle Head
7	Right of Z-Scale	27	Upper Rear of Spindle Head
8	Right Center of Z-Scale	28	Left of Top of Spindle Head
9	Left Center of Z-Scale	29	Middel of Top of Spindle Head
10	Left of Z-Scale	30	Right of Top of Spindle Head
11	Top of X-Way	31	Bottom Left of Bed
12	Bottom of X-Way	32	Top Left of Bed
13	Bottom of X-Head	33	Bottom Right of Bed
14	Top of X-Head	34	Not Used
15	Bottom of Z-Slide	35	Near X-Drive Motor Shaft Bearing
16	Top Left of Z-Slide	36	Left Z-Ballscrew Bearing
17	Bottom Right of Z-Slide	37	Right Z-Ballscrew Bearing
18	Top Right of Z-Slide	38	X-Ballscrew Housing
19	Hydraulic Tank	39	Z-Ballscrew Nut

Table 1. Thermocouple locations on the turning center.

```
        397          41           1   1.000000
POSITION  DIFF.                 0           1           2
         1  -12.70000     -1.1631000E-03   22.58143        22.58286
   22.61200
         2  -25.40000     -3.2432000E-03   22.58286        22.58571
   22.61400
         4  -50.80000     -7.4136001E-03   22.58571        22.59143
   22.61800
         5  -63.50000     -9.2676003E-03   22.58714        22.59429
   22.62000
         7  -88.90000     -1.2797900E-02   22.59000        22.60000
   22.62400
         8  -101.6000     -1.4575700E-02   22.59143        22.60286
   22.62600
        10  -127.0000     -1.7064501E-02   22.59429        22.60857
   22.63000
        11  -139.7000     -1.8334400E-02   22.59571        22.61143
   22.63200
        13  -165.1000     -2.0315200E-02   22.59857        22.61714
   22.63600
        14  -177.8000     -2.1559600E-02   22.60000        22.62000
   22.63800
        16  -203.2000     -2.3108700E-02   22.60286        22.62571
   22.64200
        17  -215.9000     -2.3184700E-02   22.60429        22.62857
   22.64400
         2  -25.40000     -4.2566000E-03   22.63571        22.71314
   22.69029
         3  -38.10000     -6.3721999E-03   22.63857        22.72971
   22.69543
         5  -63.50000     -1.0079800E-02   22.64429        22.76286
   22.70572
         6  -76.20000     -1.1552600E-02   22.64714        22.77943
   22.71086
         8  -101.6000     -1.5183800E-02   22.65286        22.81257
   22.72114
         9  -114.3000     -1.6554900E-02   22.65571        22.82914
   22.72629
        11  -139.7000     -1.8839600E-02   22.66143        22.86229
   22.73657
        12  -152.4000     -1.9727901E-02   22.66429        22.87886
   22.74171
        14  -177.8000     -2.1961600E-02   22.67000        22.91200
   22.75200
        15  -190.5000     -2.2824399E-02   22.67286        22.92857
   22.75714
        17  -215.9000     -2.3432100E-02   22.67857        22.96171
   22.76743
         1  -12.70000     -1.1750000E-03   22.73372        23.26657
   22.86857
         3  -38.10000     -5.3284001E-03   22.74114        23.27971
   22.88571
         4  -50.80000     -7.1814000E-03   22.74486        23.28629
   22.89429
         6  -76.20000     -1.0404900E-02   22.75229        23.29943
   22.91143
         7  -88.90000     -1.2283200E-02   22.75600        23.30600
   22.92000
         9  -114.3000     -1.5252300E-02   22.76343        23.31914
   22.93714
```

Table 2. A partial sample input data.

regression.

The original data set included results from four thermocouples that failed. They were thermocouples numbered 3, 4, 16, and 19. These readings were set to zero and left in the neural net training process to determine whether the neural net algorithm would weight the fitting towards the nonzero columns. The results of the training are given in the next section.

After 1000 iterations of the conjugate gradient optimization routine the residuals between the target values and the predicted values exhibited a normal distribution. The following statistics summarize the distribution of residuals:

Mean = 0.4791046E-5 mm
Std. Dev. = 0.3068930E-3 mm
Range = 0.2006001E-2 mm
Minimum = -0.1096E-2 mm
Maximum = 0.9100009E-3 mm

A histogram of the residuals is shown in Figure 6 . The figure shows the near normality of the distribution with a slight skew to the positive side. The normality of these residuals is a sign that the errors after the fit exhibit randomness and do not reflect errors in the model.

Two other tests of the normality of the residuals are shown in Figures 7 and 8. The first is the normality plot. The normality plot for a given distribution is a graphical data analysis technique for determining if the given distribution provides a good distributional fit to the data. The vertical axis is the ordered raw data. The horizontal axis is the ordered statistic from the normal distribution with mean 0 and standard deviation 1. The axis shows from -3 standard deviations to +3 standard deviations. The linearity of the plot is of interest. The more linear the plot the better the distributional fit. The plot exhibits a linear trend, thus indicating normality.

Figure 8 is a correlation coefficient plot. It is a graphical data analysis technique for determining which member of an entire family of distributions provides the best fit to the data. The normal distribution is one of a large family of distributions called the Tukey lambda distributional family. The family is parameterized by a simple value extending from -2 to 2. The value of the family representing the normal distribution is 0.14. Thus if the data exhibits a normal distribution, the

correlation coefficient with each of the members of the Tukey family should reach a maximum with respect to the family parameter at 0.14. In fact the residual data achieves a maximum of 0.9911880 at 0.14. This shows how close the residual data correlates with the normal distribution.

These results were generated by the software package DATAPLOT (see Filliben [19]) at NIST.

These fitting results show that the neural net algorithm does take the zero columns as part of the input pattern and still reduces the regularized RMS error so that the standard deviation of the residual errors falls within the desired value of less than half a micrometer. In particular, the standard deviation is approximately 0.3 micrometers with a mean of approximately 0.005 micrometers.

After fitting the neural net, the optimum weights were then used to compute the predicted output for 198 test samples of the X-Displacement (Down) data. These samples were all selected separately from the pattern samples used to fit the weights. The results were as follows:

Mean = -0.2961085 E - 4 mm
Std. Dev. = 0.2255760 E - 3 mm
Range = 0.2019778 E - 2 mm
Minimum = -0.1119867 E - 2 mm
Maximum = 0.8999112 E - 3 mm

These are comparable to the results obtained during the fitting process. They again fall within the desired criteria even through the columns with zeroed data have been left as part of the patterns. The Kolmogorov-Smirnov statistic for the residual distribution does not confirm that it is normally distributed. However, Figure 9 shows that the major position of the distribution is centered near zero as the mean suggests and falls between -5E-3 and +5E-3. This statistical test is thus too fine a test for the distribution generated. Although its results are reported, they will not be referred to in the later neural net fitting trials.

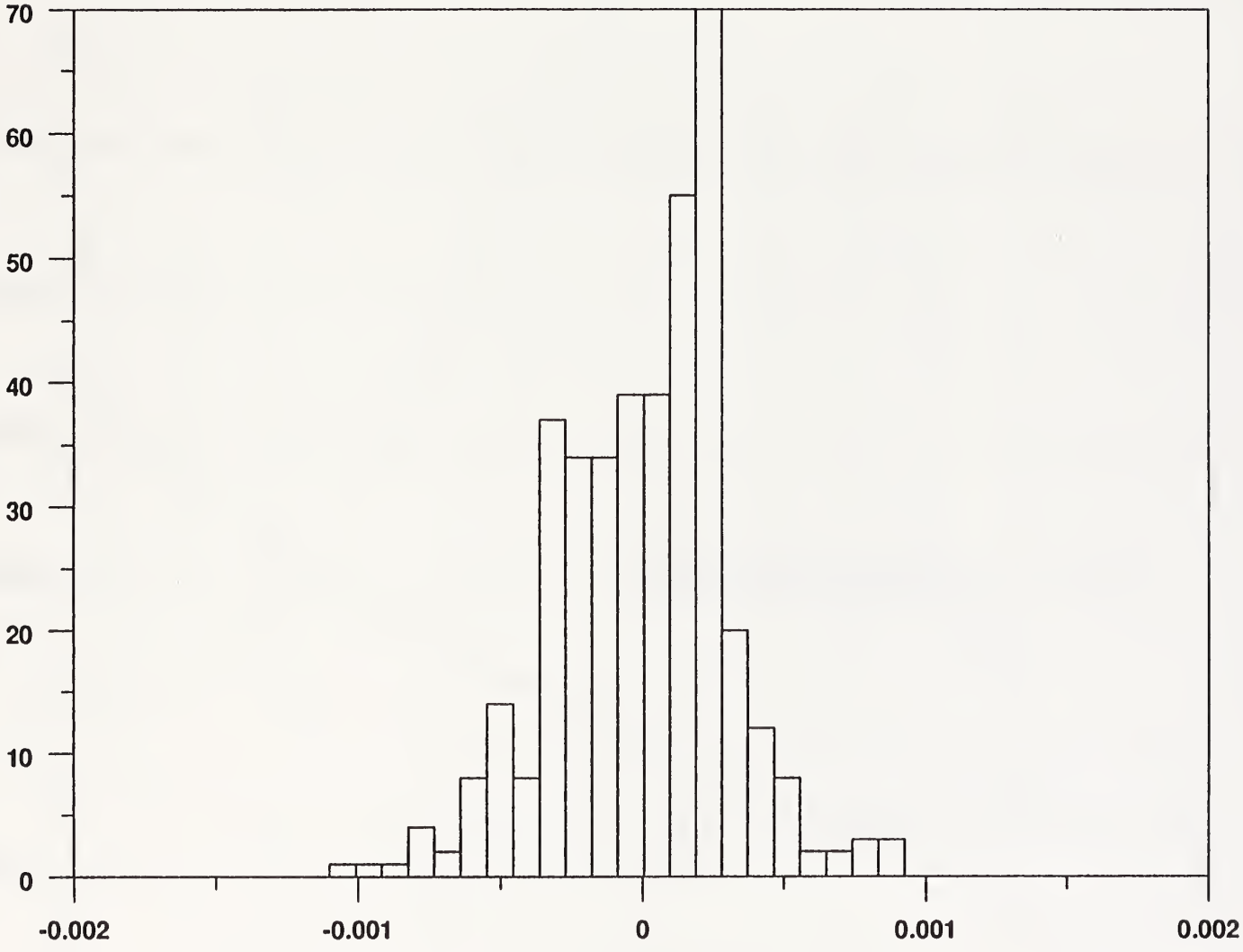

Figure 6. Histogram plot of the residual error distribution after the fit of the x-displacement (down) data in mm.

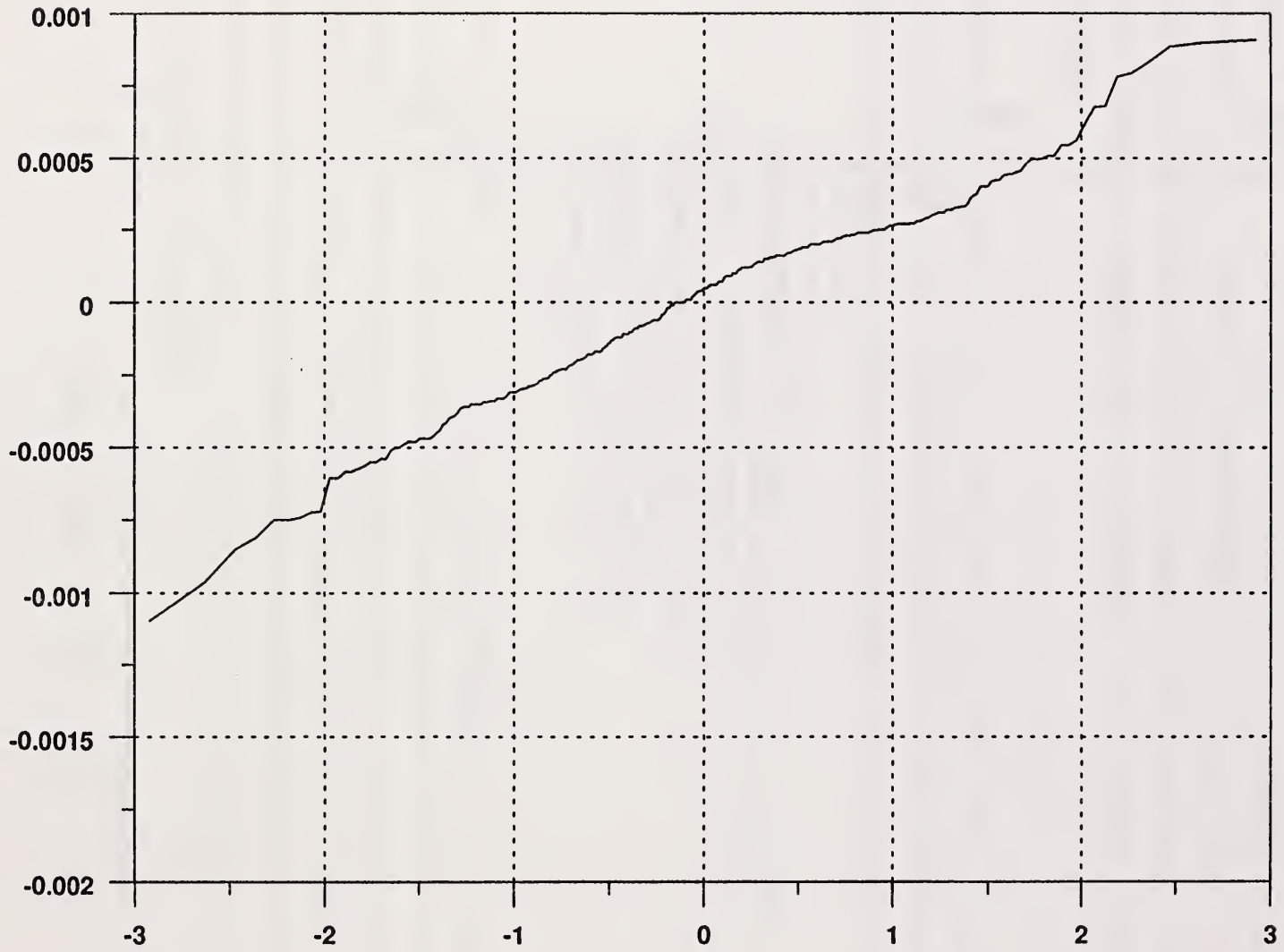

Figure 7. Normality plot for residuals of the x-displacement (down) data.

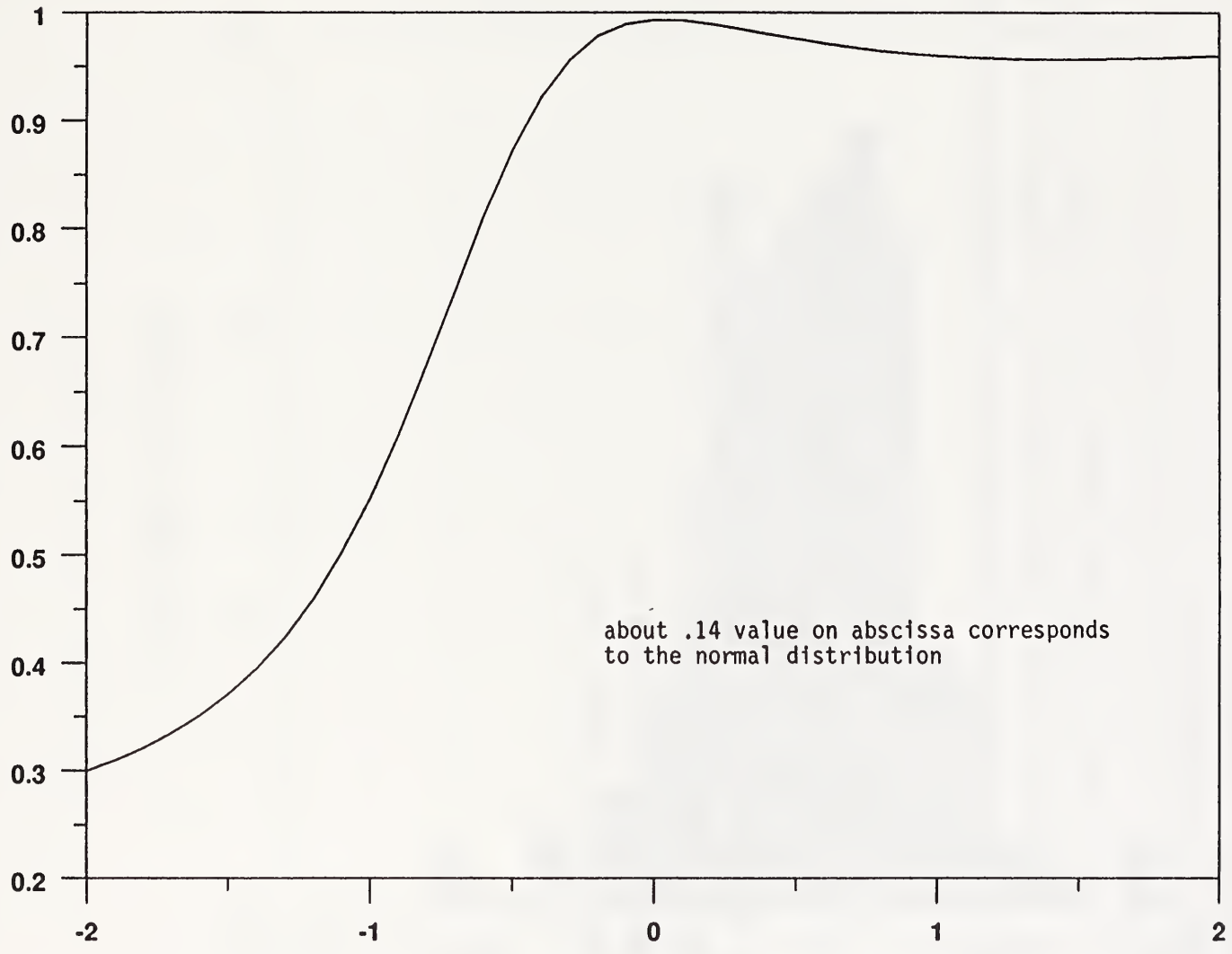

Figure 8. Correlation Coefficient Plot for the residuals of the x-displacement (down) data

Another training run of 1500 iterations of the neural net was done but without the zeroed data from the four failed theromcouples. There were then 37 input nodes. The results of this fit are

Mean = 0.2527817 E - 5
Standard Deviation - 0.1701827 E - 3
Minimum = -0.6710416 E - 3
Maximum = 0.7603451 E - 3

These values show that the neural net algorithm was able to reduce the standard deviation of the error well below the accpetable criteria. Figure 10 shows the error reduction curve for the regularized RMS error as a function of the iteation cycles. Figure 11 shows the histogram of the residual errors.

A test was performed with 198 patterns. The residual statistics for this test were:

Mean = -0.2718773 E - 4 mm
Standard Deviation 0.1935153 E - 3 mm
Minimum = -0.8775685 E - 3 mm
Maximum = 0.1151932 E - 2 mm

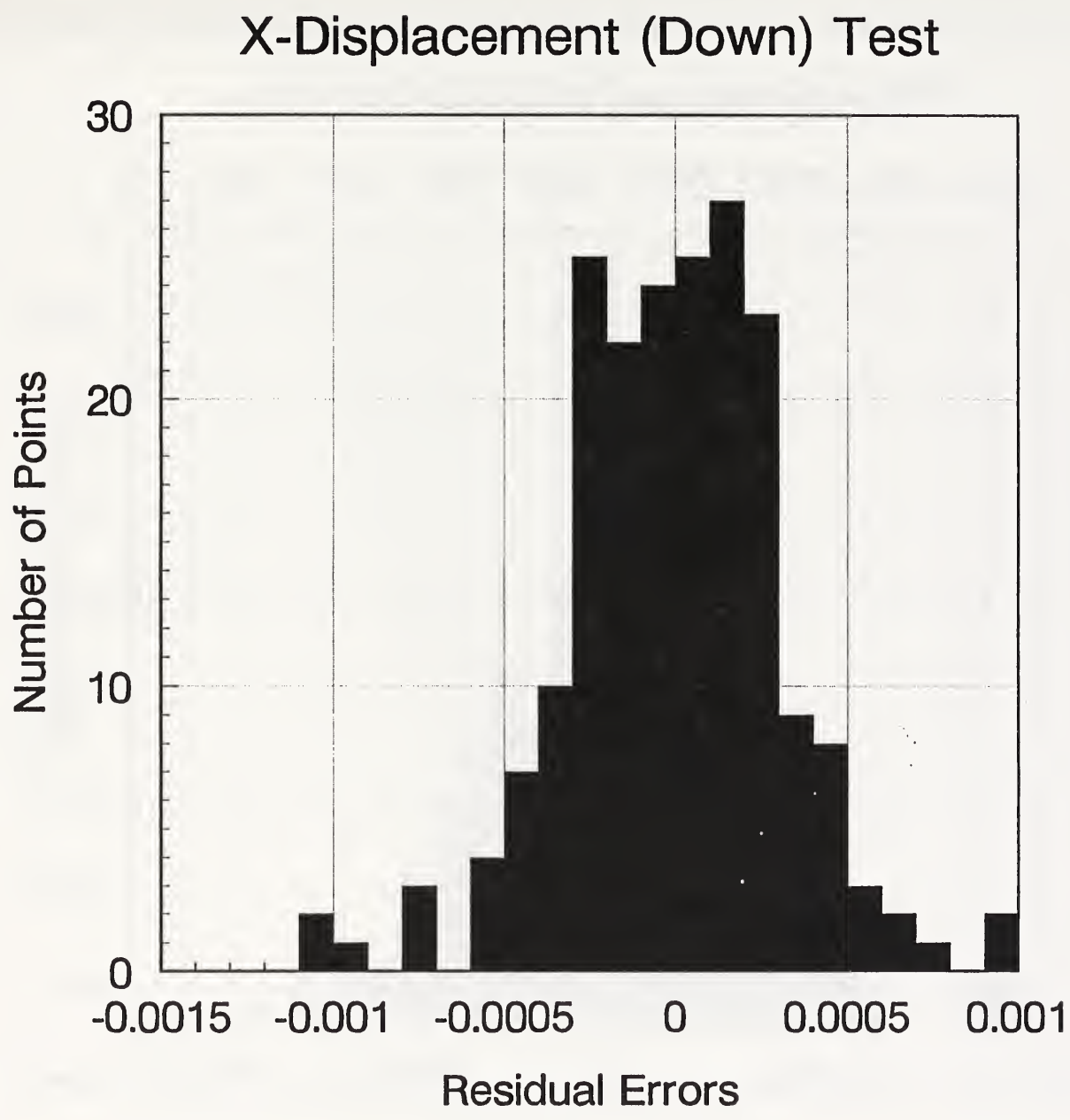

Figure 9. Histogram plot of the residual error distribution after the test of the x-displacement (down) data. All data columns used. Residual errors are in millimeters.

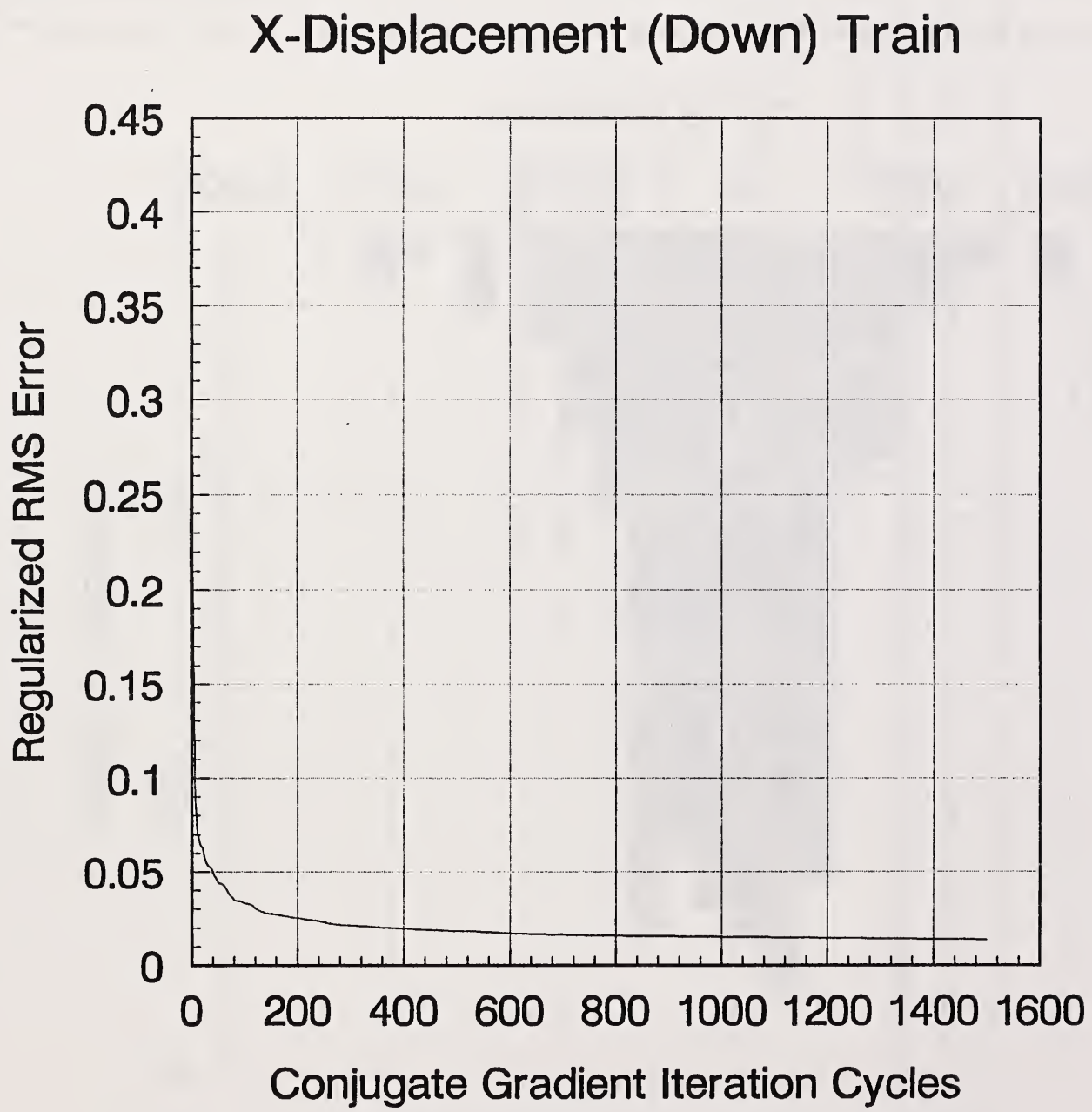

Figure 10. Error reduction curve for training on data with failed thermocouples removed. Regularized RMS errors are in millimeters.

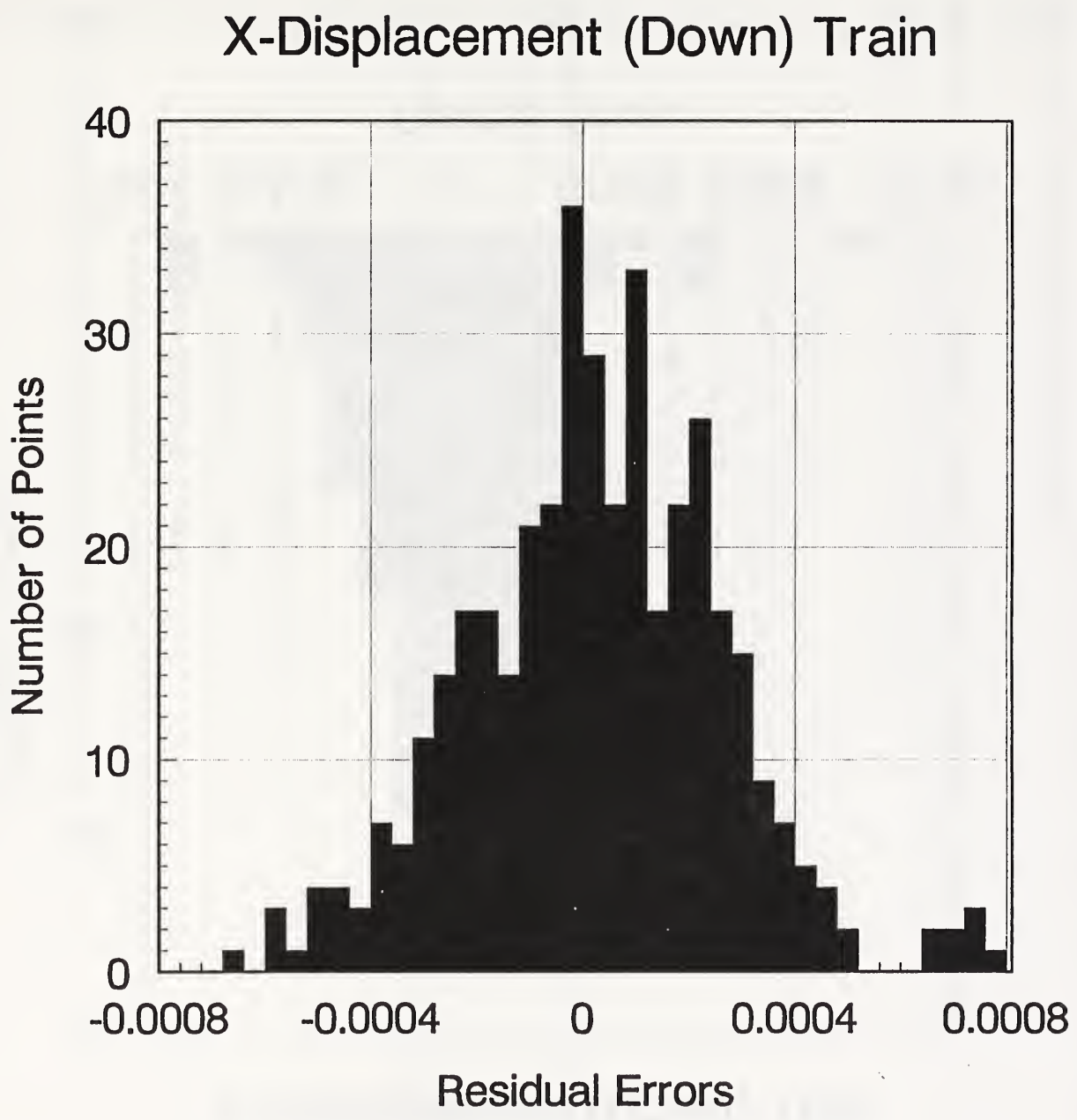

Figure 11. Histogram of residuals for training data with failed thermocouples removed. Residual errors are in millimeters.

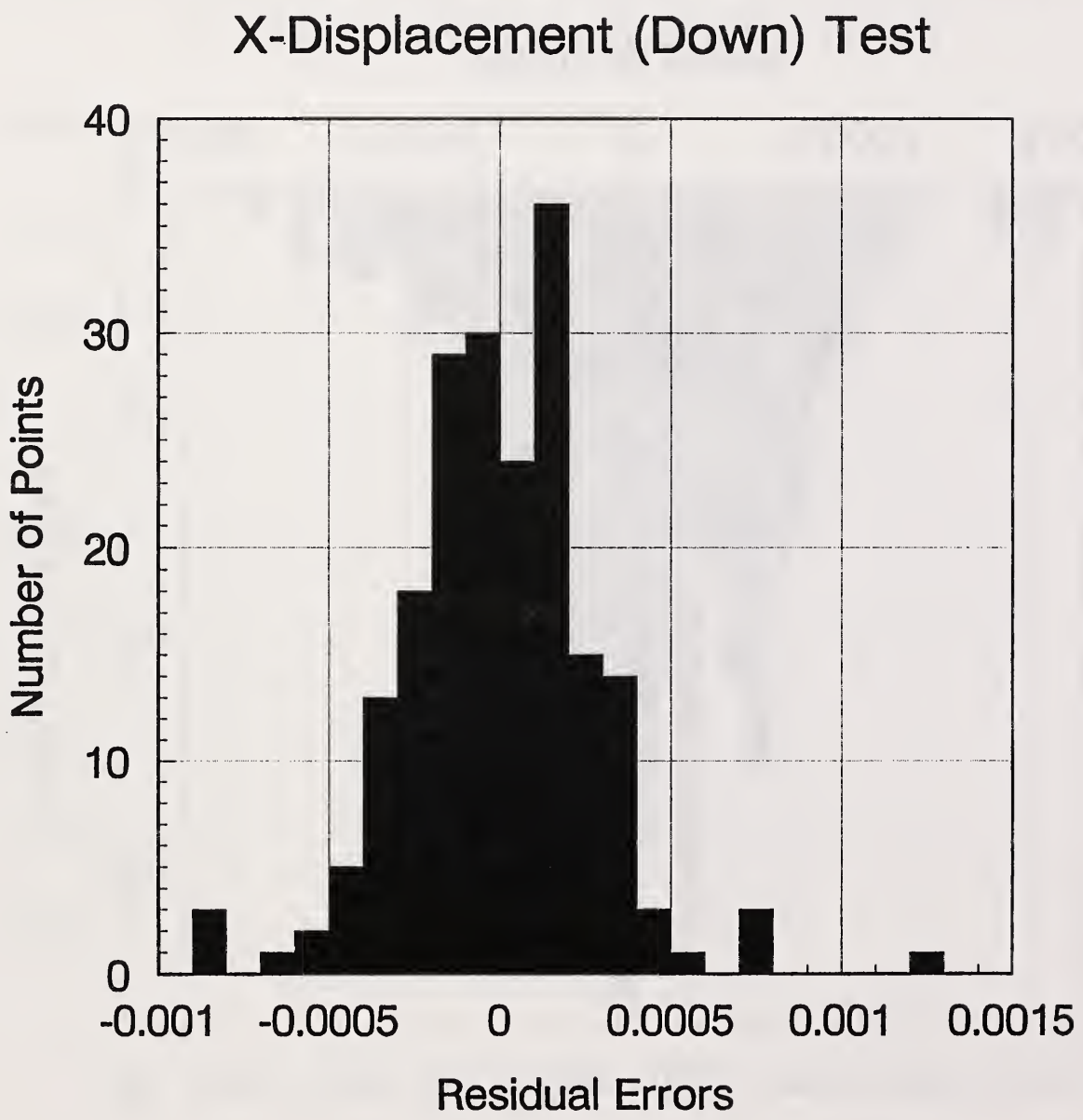

Figure 11A. Histogram of residuals for testing the trained neural net with failed thermocouples removed. Residual errors are in millimeters.

9.2 X-Displacement (Up)

In this data set all columns of thermocouple data that were not usable were eliminated from the fitting process. Only 35 thermocouples were operating since thermocouples 0, 4, 5, 17, and 20 failed to function. Therefore, 36 inputs to the neural net were used with 60 hidden nodes and one output node. This generated 2581 unknown weights.

After 1500 iterations the conjugate gradient algorithm reported the following statistics for the residual error between the measured and predicted outputs:

Mean = 0.1994597 E - 5 mm
Standards Deviation = 0.1338334 E - 3 mm
Minimum = -0.1036018 E - 2 mm
Maximum = 0.7032966 E - 3 mm

The regularized RMS error as a function of the iterations cycle is shown in Figure 12 and the histogram of the residuals after the fitting is shown in Figure 13.

A test was run on independent data using the fitted weights and the residual errors reported were:

Mean = 0.2145889 E - 4 mm
Standard Deviation = 0.1509593 E - 3 mm
Minimum = -0.8786396 E - 3 mm
Maximum = 0.7763319 E - 3 mm

The histogram of the residuals for the test run are shwon in Figure 13a.

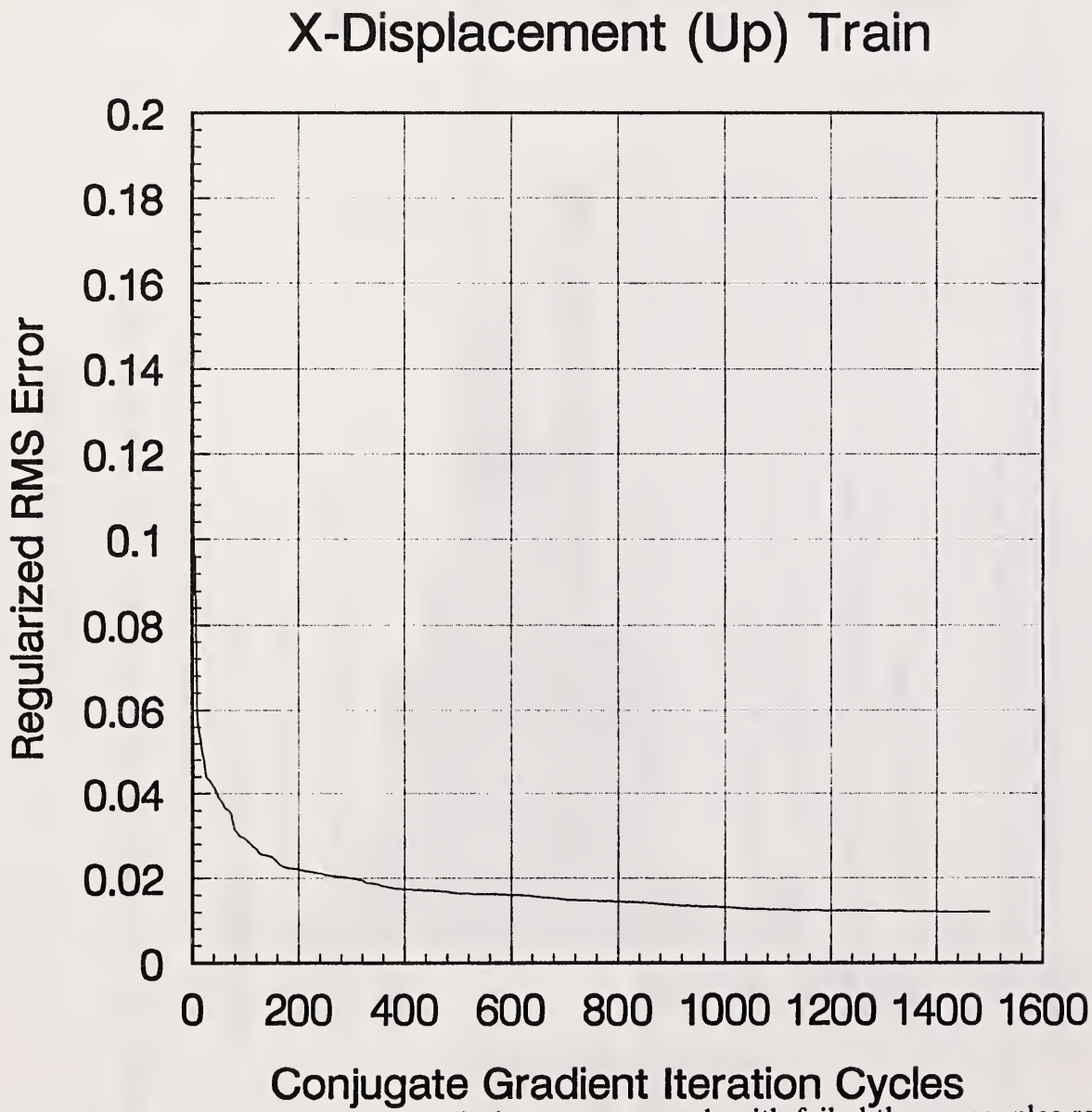

Figure 12. Regularized RMS error training curve for X-displacement upwards with failed thermocouples removed. Regularized RMS errors are in millimeters.

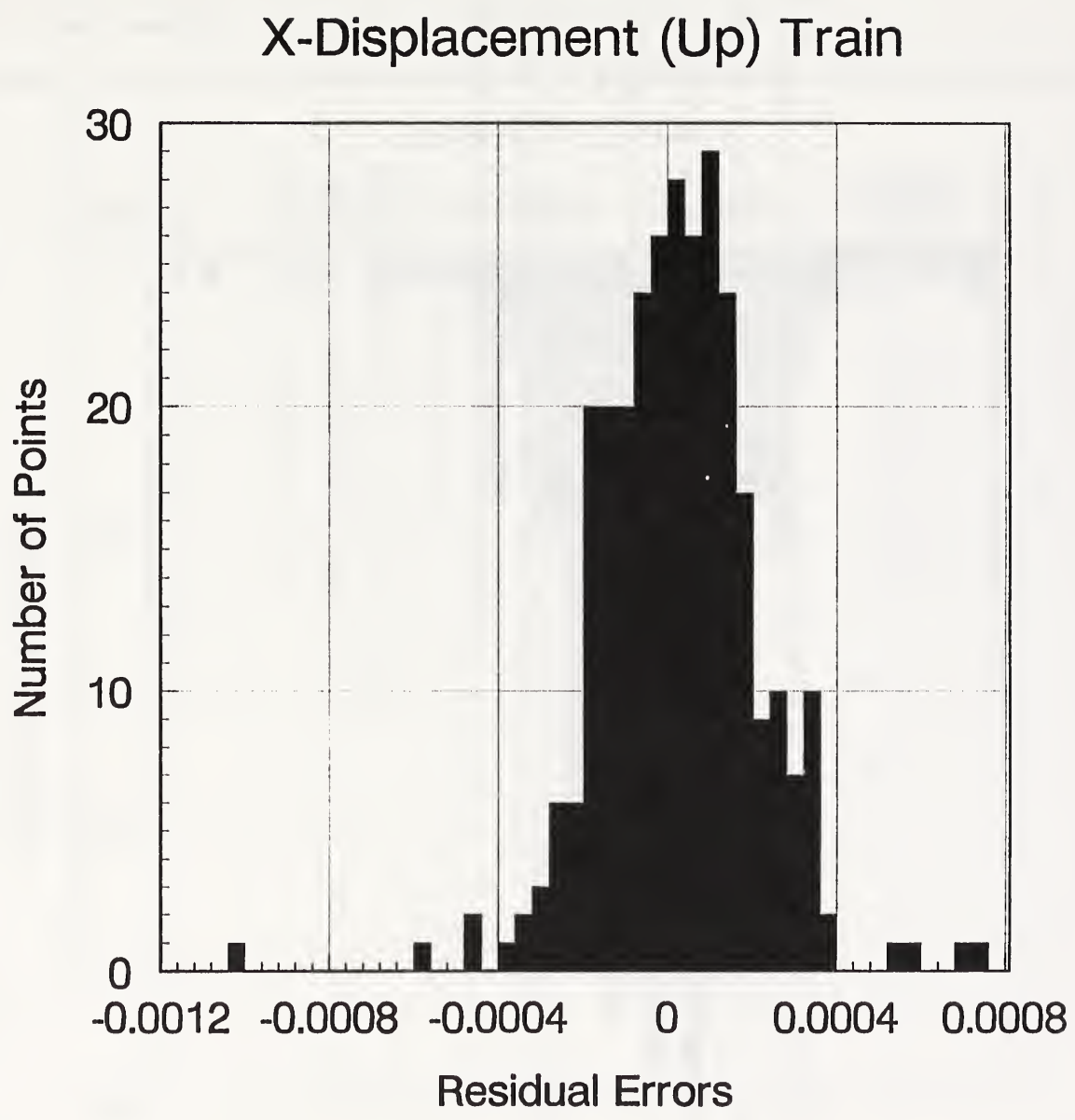

Figure 13. Histogram of residuals after neural net training on X-displacement upward data with failed thermocouples removed. Residual errors are in millimeters.

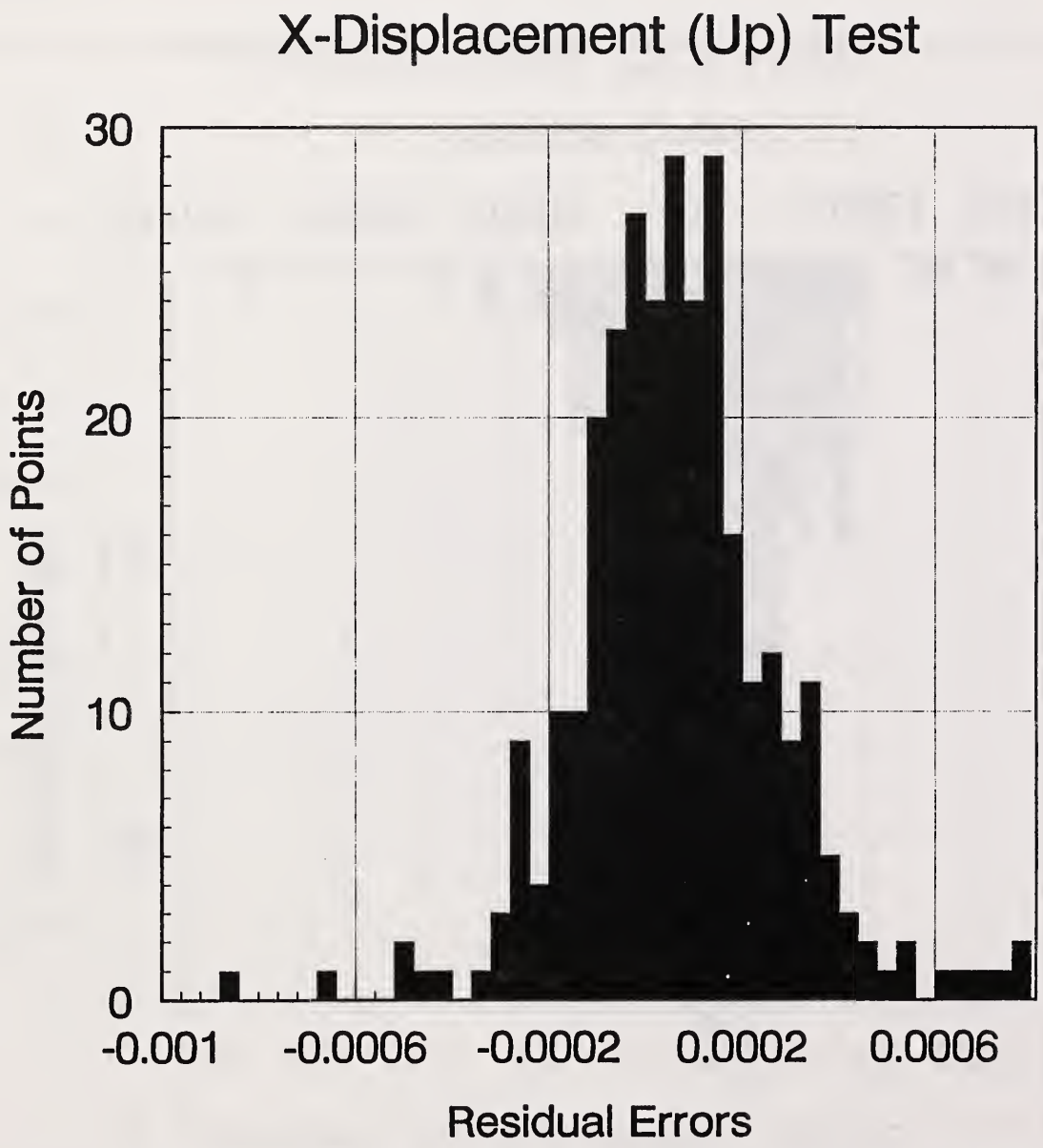

Figure 13A. Histogram of residuals after test data applied to neural net for X-displacement. Upwards failed thermocouples removed. Residual errors are in millimeters.

9.3 X-Displacement (Down) with X-Axis Related Thermocouples

In this numerical experiment only X-Axis related thermocouples as shown in Table 1 are used for inputs. The thermocouples used in the fitting of the data were numbered 0, 1, 11, 12, 13, 14, 35, and 35 in Table 1. The fitting algorithm reported the following results:

Mean = 0.1879598 E - 5 mm

Standard Deviation = 0.3769576 E - 3 mm

Minimum = -0.2565682 E - 2 mm

Maximum = 0.1568761 E - 2 mm

The regularized RMS error curve is shown in Figure 14 and the histogram of residual errors is given in Figure 15. Note that the residual errors in this case seem to distribute themselves bimodally around zero. The base of the distribution is broader than those resulting from fitting experiments using all the usable thermocouples.

A test was then performed on 198 data patterns not used in the fitting process. The statistics associated with the residuals for this test data were:

Mean = -0.2894112 E - 4 mm

Standard Deviation = 0.3918001 E - 3 mm

Minimum = -0.2590802 E - 2 mm

Maximum = 0.1427975 E - 2 mm

The histogram of the test residuals is given in Figure 15A.

9.4 X-Displacement (Down) with Z-Axis Related Thermocouples

In this section only the Z-Axis related thermocouples are used as inputs. From Table 1 the thermocouples used were 7, 8,9, 10, 15, 17, 18, 20, 21, 22, 23, 36, 37, and 39. Although 16 is z-axis related, it failed and was not used. The fitting algorithm reported the following results:

Mean = 0.9919781 E - 6 mm

Standard Deviation = 0.1879221 E - 3 mm

Minimum = -0.1144482 E - 2 mm

Maximum = 0.6796103 E - 3 mm

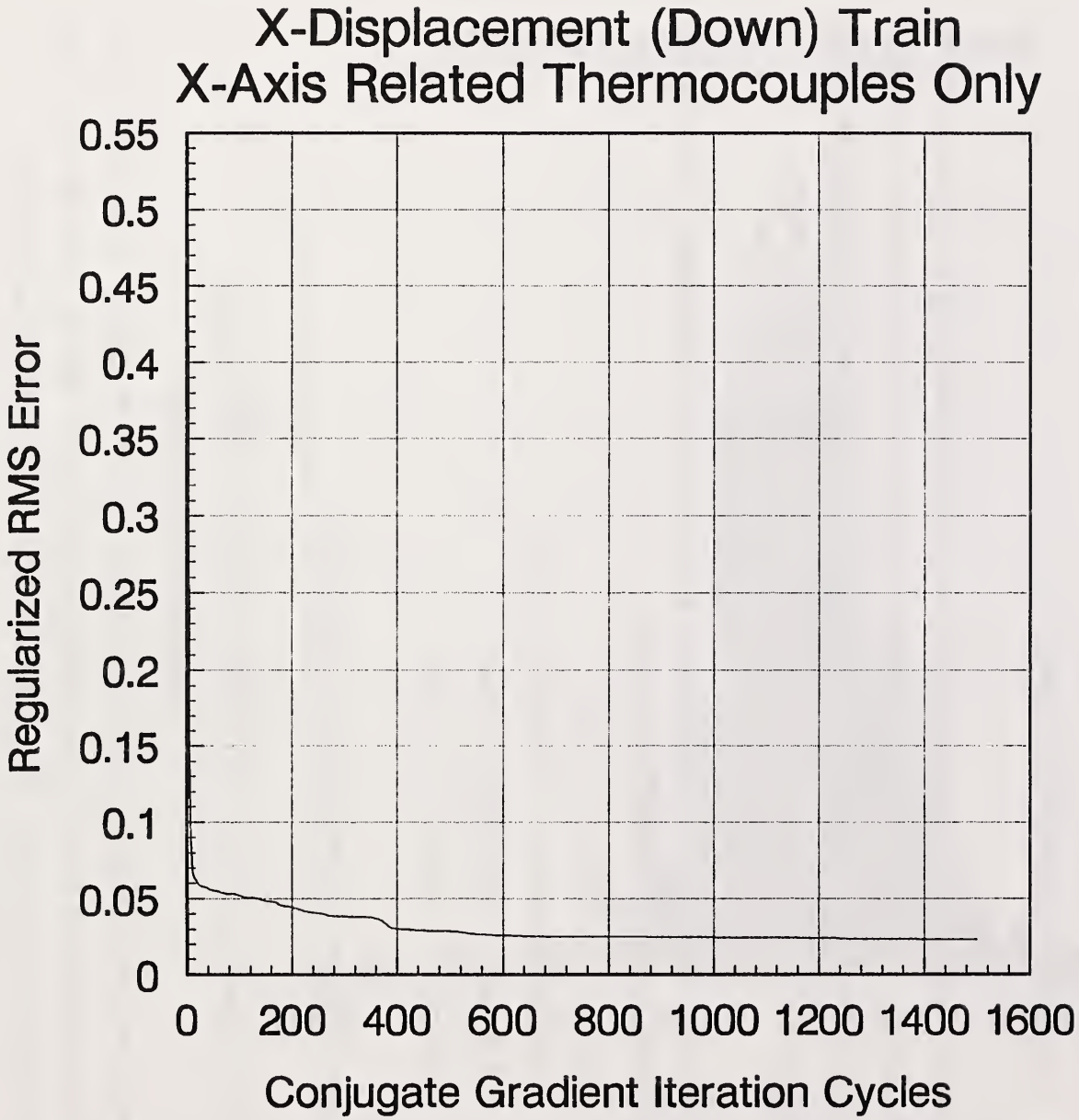

Figure 14. Regularized RMS error training curve for fitting X-displacement downwards data against X-axis thermocouples. Regularized RMS errors are in millimeters.

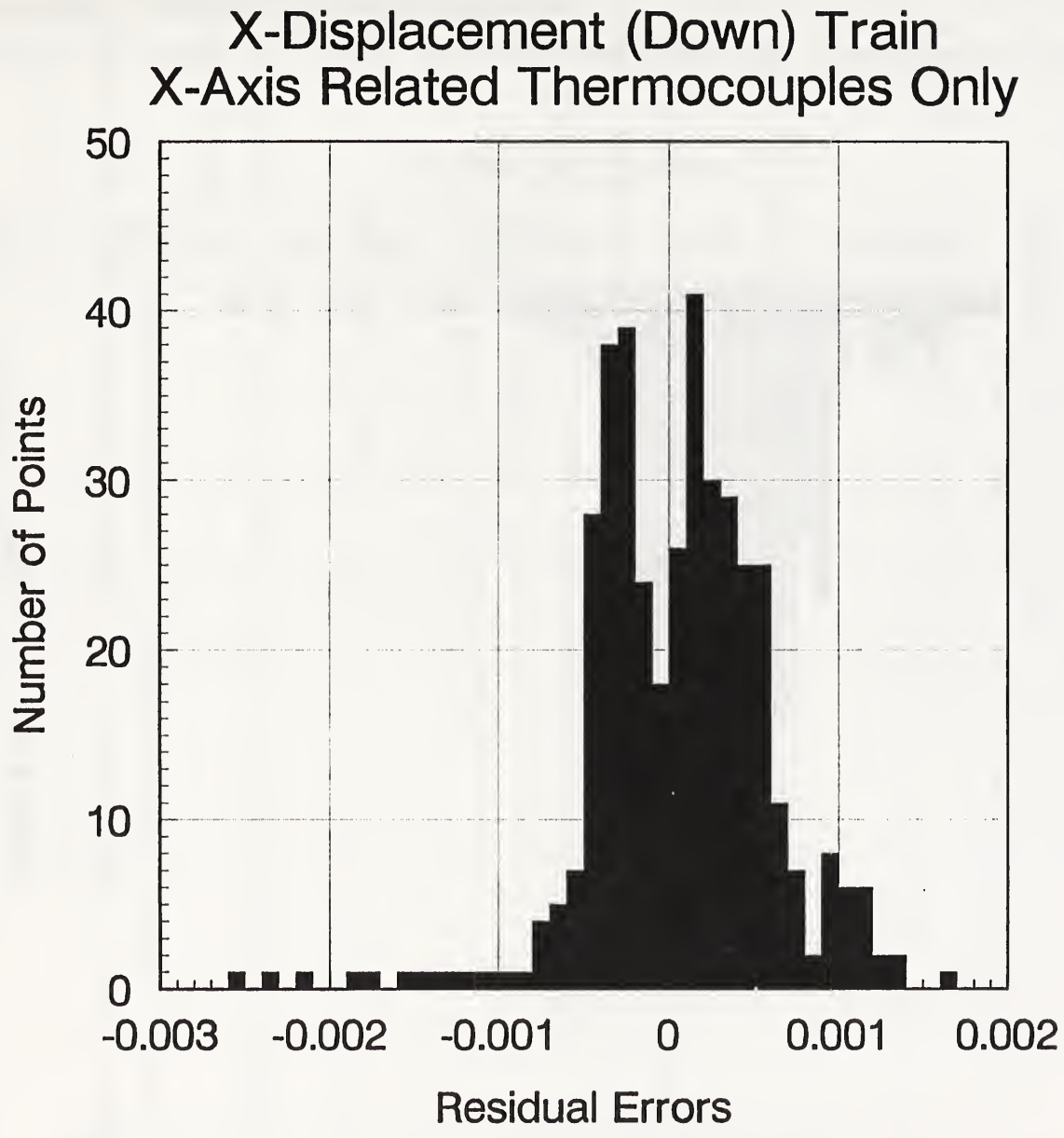

Figure 15. Histogram of residuals after training with X-displacement downwards data for X-axis related thermocouples. Residual errors are in millimeters.

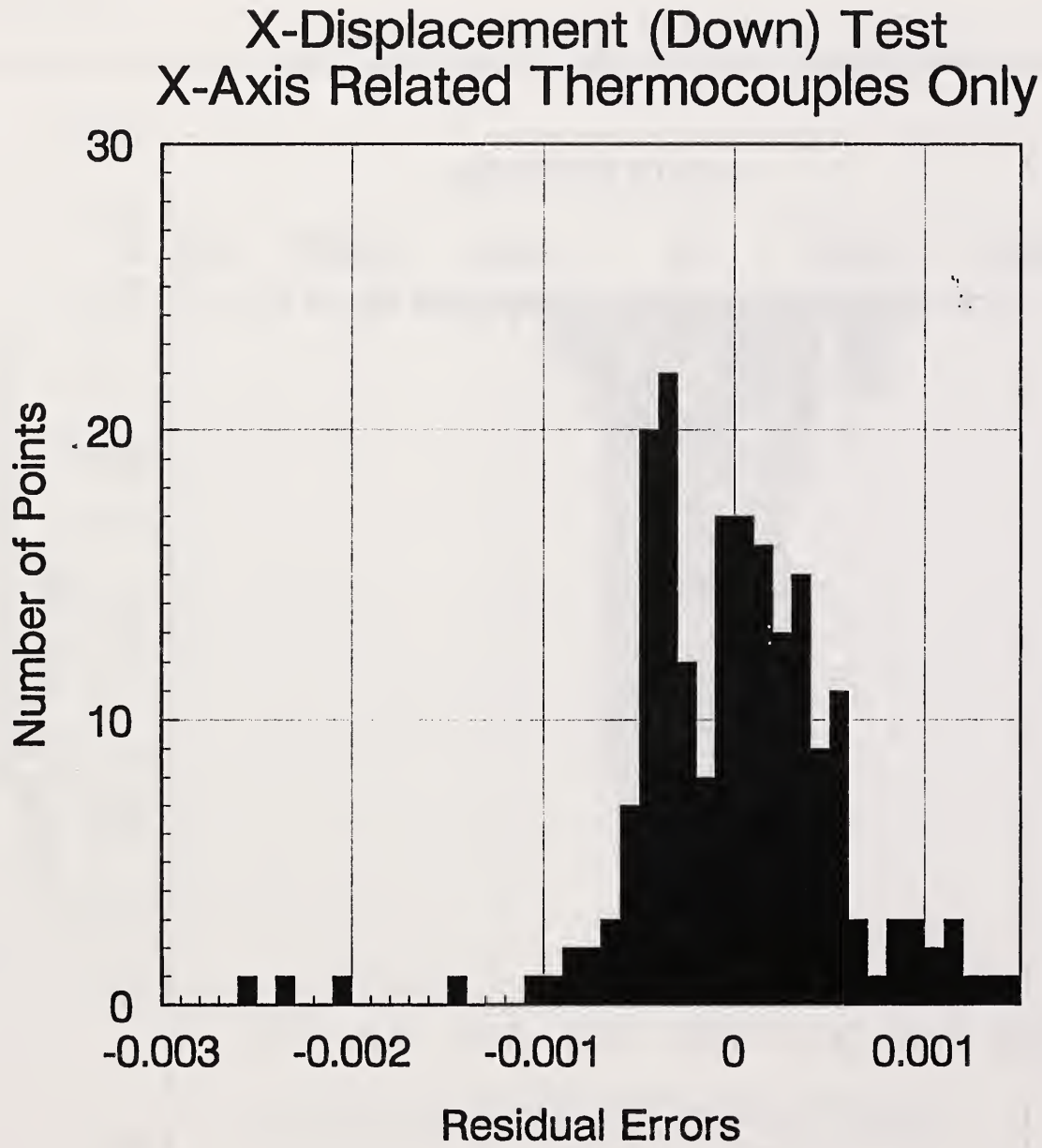

Figure 15A. Histogram of residuals for testing the trained neural net with X-displacement downward data against X-related thermocouples. Residual errors are in millimeters.

The regularized RMS error curve is shown in Figure 16 and the histogram of the residual errors is given in Figure 17.

A test was performed on 198 separate data patterns. The test statistics of the residuals were:

Mean = -0.3042790 E - 4 mm
Standard Deviation = 0.2007157 E - 3 mm
Minimum = -0.1336623 E - 2 mm
Maximum = 0.9711841 E - 3 mm

A histogram of the test residuals is given in Figure 17A.

9.5 X-Displacement (Up) with X-Axis Related Thermocouples

This numerical experiment repeats that described in section 9.3 but used the data measured while the tool turret moved upwards. Seven thermocouples were used. They were 1, 11, 12, 13, 14, 35, and 38 as shown in Table 1. The fitting algorithm reported the following results:

Mean = 0.5959608 E - 5 mm
Standard Deviation = 0.2965477 E - 3 mm
Minimum = -0.1392284 E - 2 mm
Maximum = 0.124761 E - 2 mm

The regularized RMS error curve is shown in Figure 18 and the histogram of residual errors is given in Figure 19.

A test of the fitted weights was performed on 297 patterns of data. The test statistics of the residuals were given as:

Mean = 0.2316002 E - 4 mm
Standard Deviation = 0.2984127 E - 3 mm
Minimum = -0.1407236 E - 2 mm
Maximum = 0.1203012 E - 2 mm

A histogram of the residuals is given in Figure 19A.

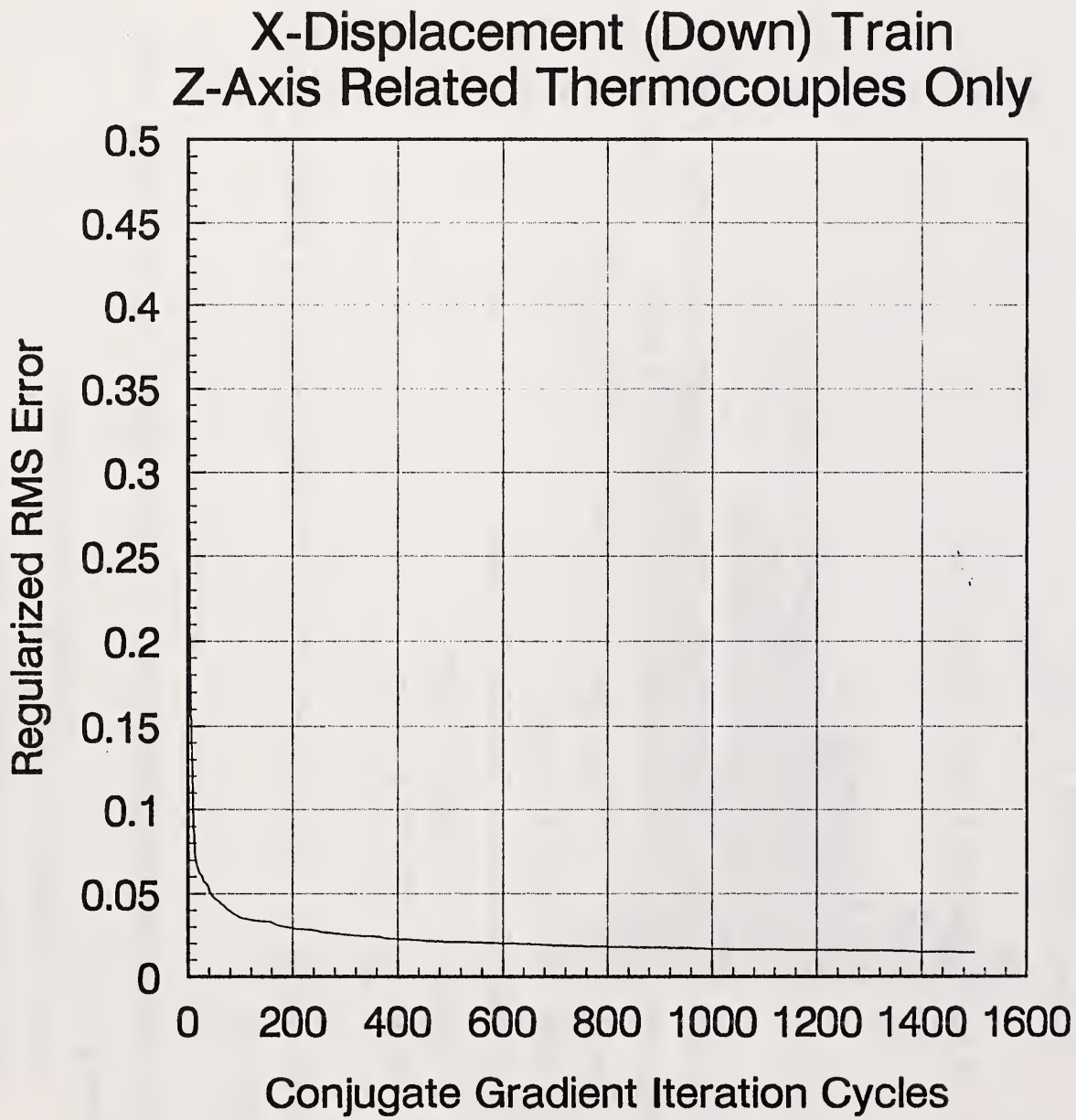

Figure 16. Regularized RMS error training curve for X-displacement downwards data against Z-axis related thermocouples. Regularized RMS errors are in millimeters.

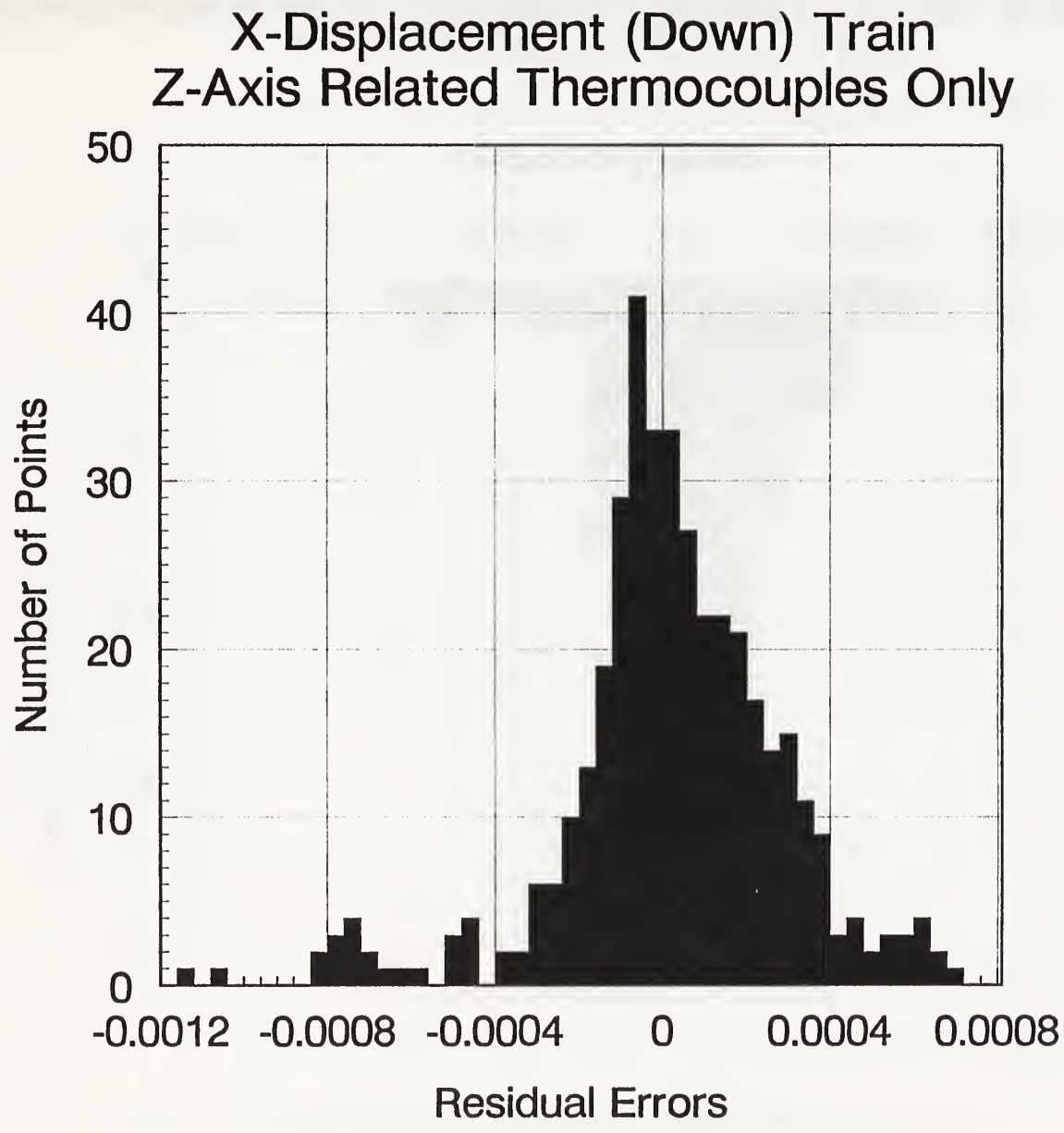

Figure 17. Histogram of residuals after training of X-displacement downwards data against Z-axis related thermocouples. Residual errors are in millimeters.

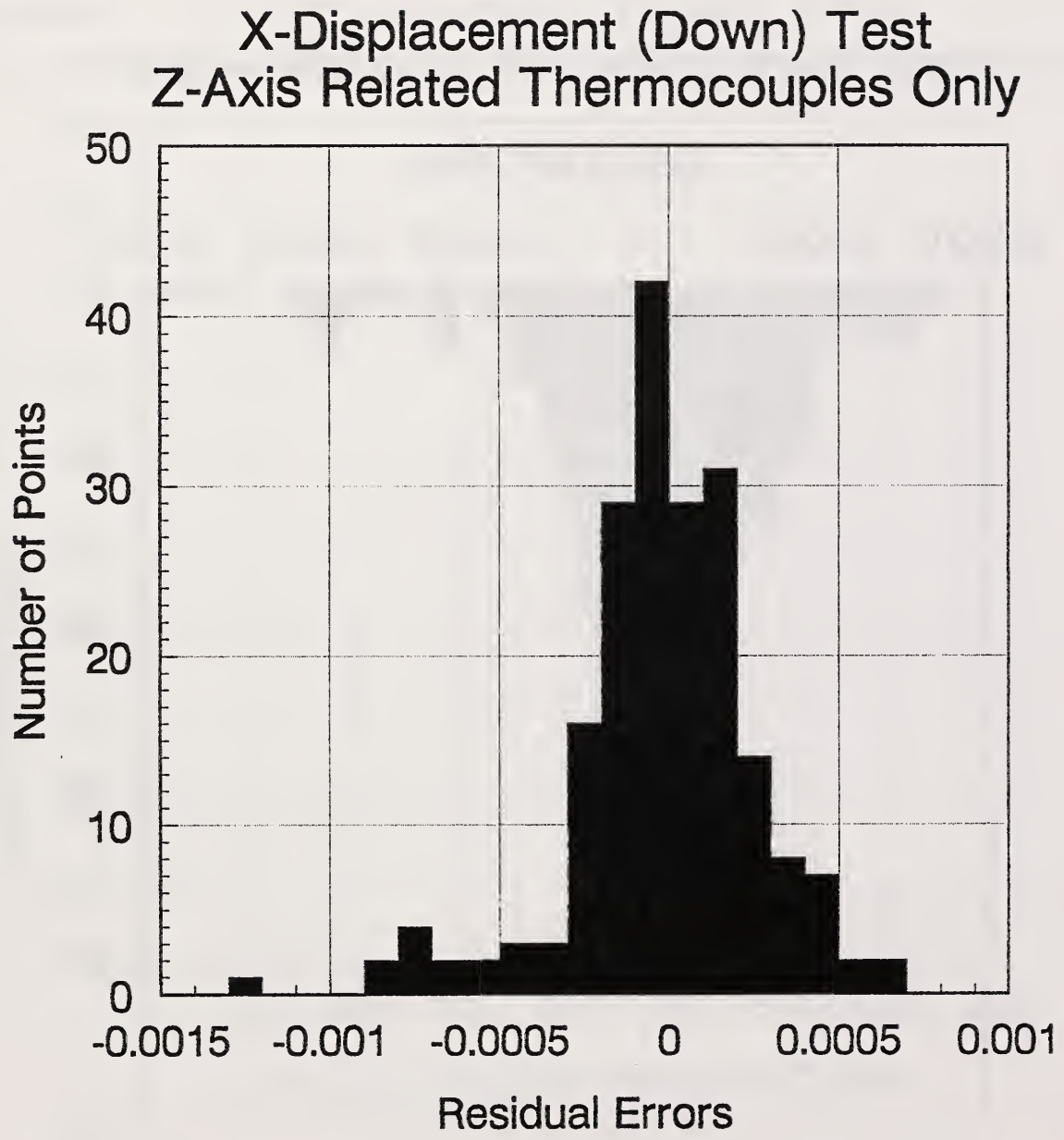

Figure 17A. Histogram of residuals for test data for X-displacement downwards against Z-axis. Residual errors are in millimeters.

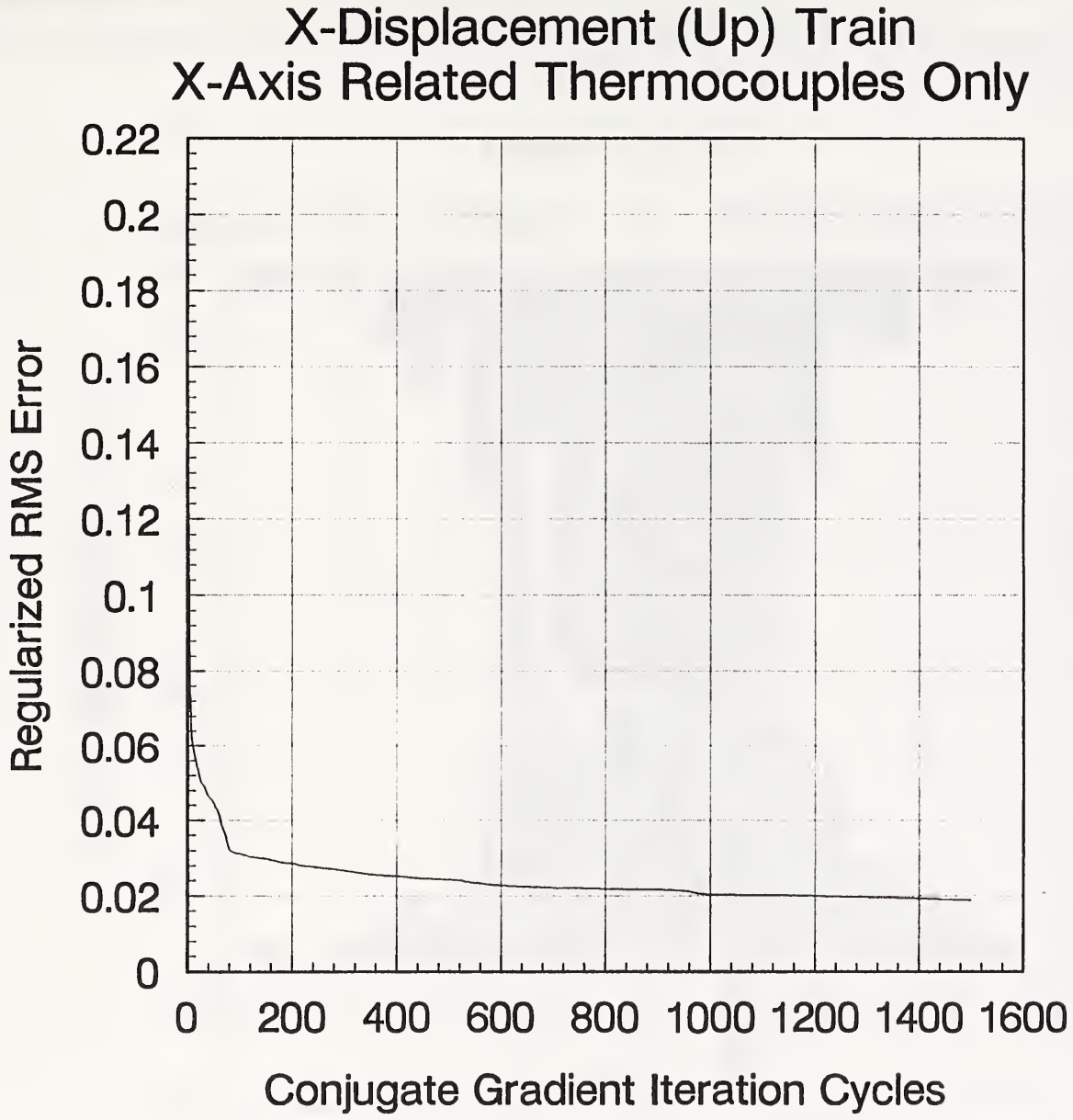

Figure 18. Regularized RMS error training curve for X-displacement upward data against X-axis related thermocouples. Regularized RMS errors are in millimeters.

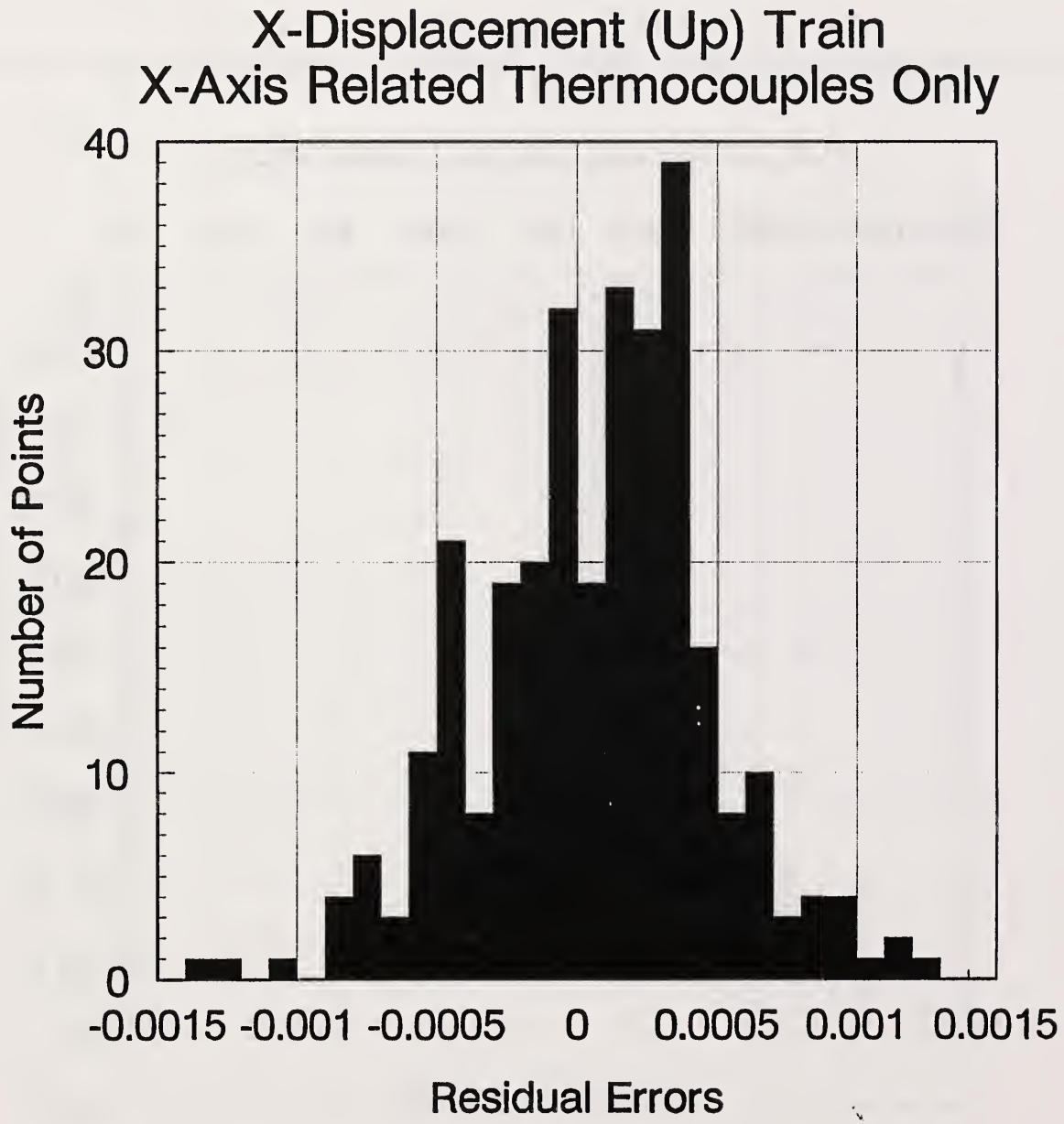

Figure 19. Histogram of residuals for training data of X-displacement upwards data against X-axis related thermocouples. Residual errors are in millimeters.

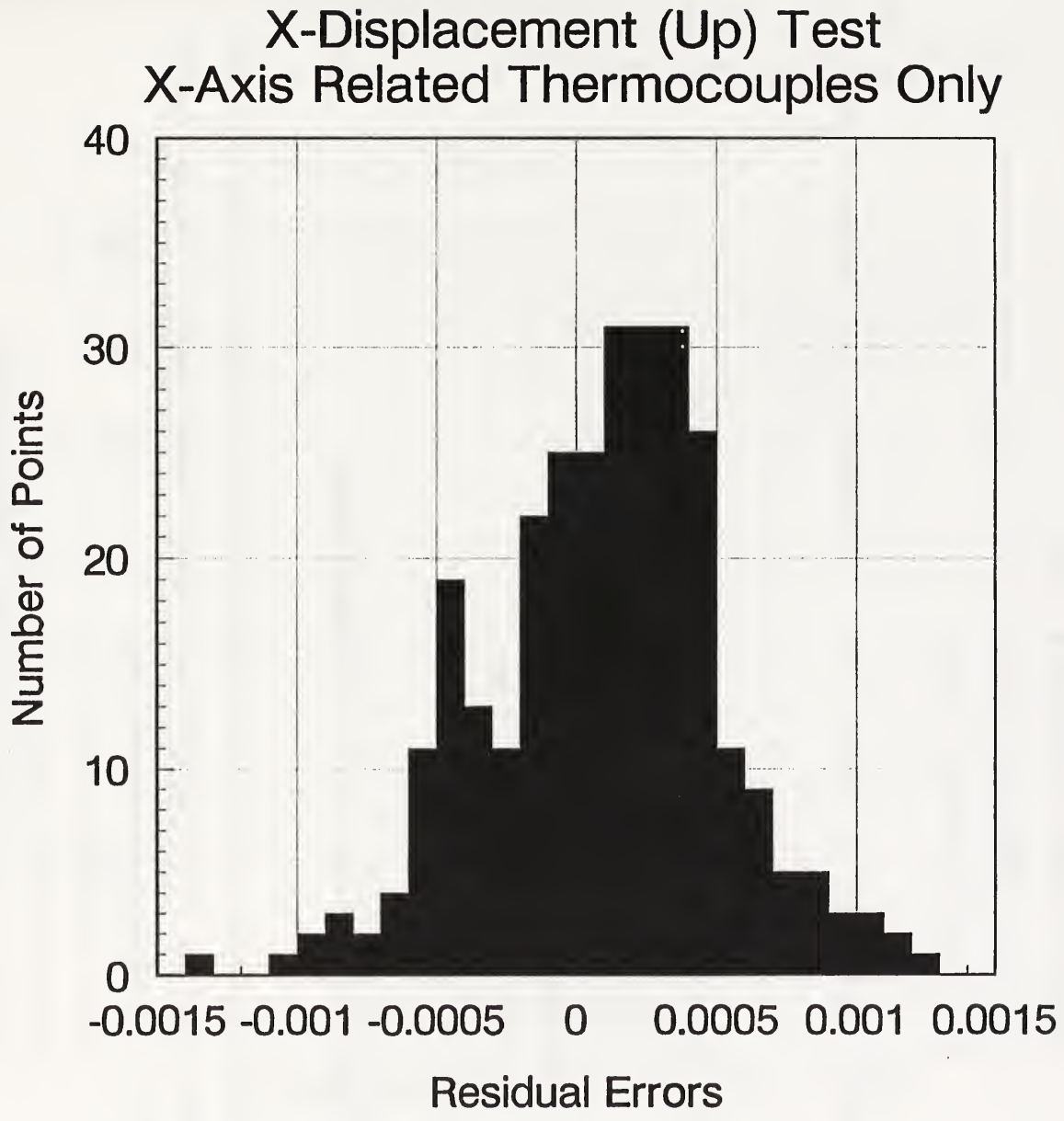

Figure 19A. Histogram of residuals of test data of X-displacement upwards against X-axis related thermocouples. Residual errors are in millimeters.

9.6 X-Displacement (Up) with Z-Axis Related Thermocouples

This numerical experiment is similar to that described in section 9.4 except that data taken while the tool turret moved upwards was used. In this case, thermocouples numbered 7, 8, 9, 10, 15, 18, 21, 22, 23, 36, 37, and 39 were used (see Table 1). The fitting results reported were:

Mean = 0.1878900 E - 6 mm

Standard Deviation = 0.3308610 E - 3 mm

Minimum = -0.2056841 E - 2 mm

Maximum = 0.1320060 E - 2 mm

The regularized RMS errors are shown in Figure 20 and the histogram of residuals is shown in Figure 21.

A test was performed on 297 data patterns. The test statistics were:

Mean = 0.1977327 E - 4 mm

Standard Deviation = 0.3356497 E - 3 mm

Minimum = -0.1775119 E - 2 mm

Maximum = 0.1378149 E - 2 mm

A histogram of the residuals is given in Figure 21A.

9.7 X-Displacement (Down) with 50 Hidden Nodes

This numerical experiment uses 36 thermocouples but reduces the number of hidden nodes used. This in effect reduces the number of logistic function basis functions used in the assumed function representation. The fitting results reported were:

Mean = 0.8658474 E - 6 mm

Standard Deviation = 0.2031533 E - 3 mm

Minimum = -0.1155412 E - 2 mm

Maximum = 0.1089299 E - 2 mm

Figure 22 shows the regularized RMS error and Figure 23 shows the distribution of residual errors. The significance of the results here is that a reduction from 60 hidden nodes to 50 hidden nodes for this data set did not change the residual distribution in any great degree.

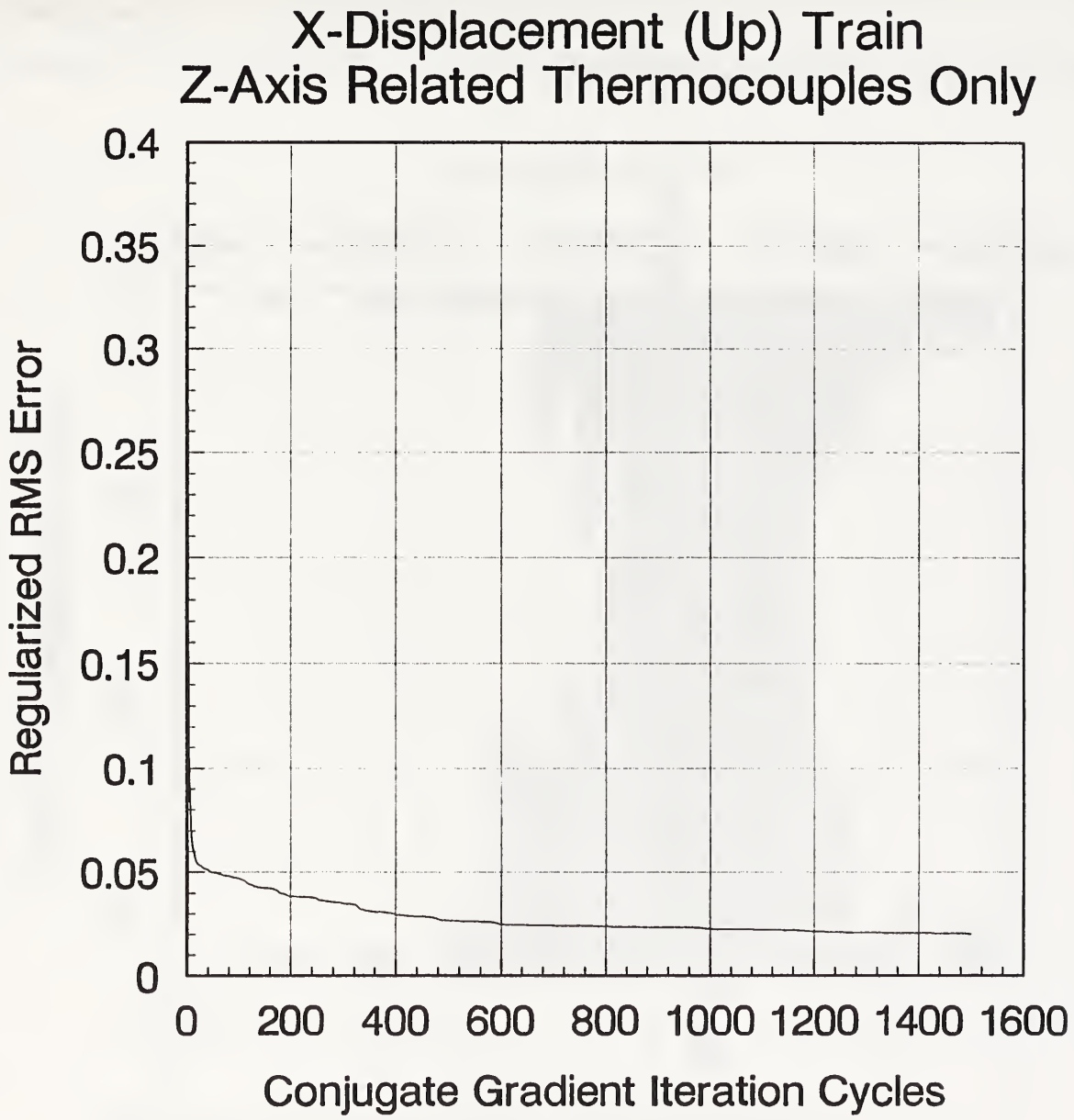

Figure 20. Regularized RMS error training curve for X-displacement upwards data against Z-axis related thermocouples. Regularized RMS errors are in millimeters.

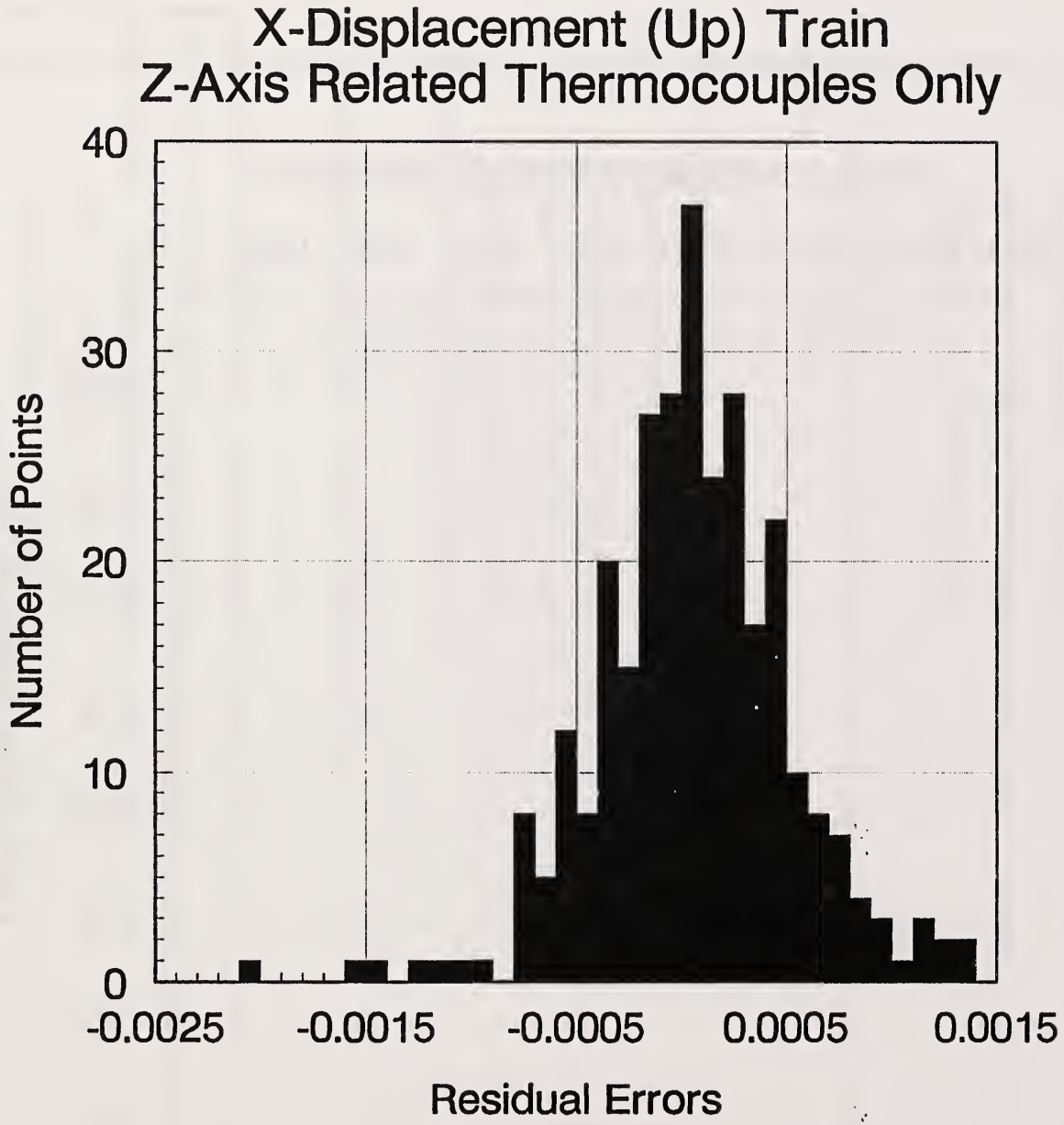

Figure 21. Histogram of residual errors for training of X-displacement upwards data against Z-axis related thermocouples. Residual errors are in millimeters.

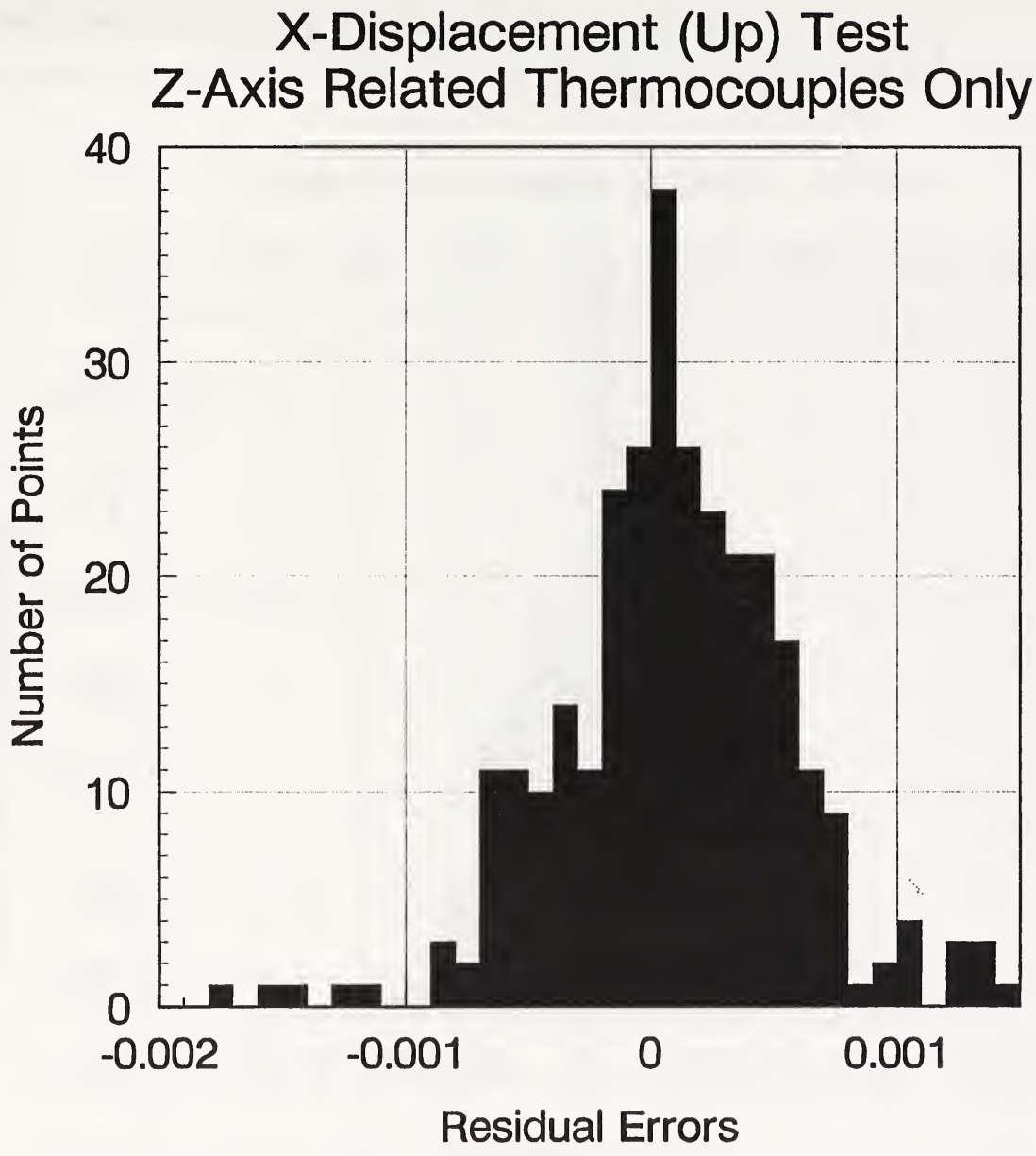

Figure 21A. Histogram of residual errors for testing X-displacement upwards data against Z-axis related thermocouples. Residual errors are in millimeters.

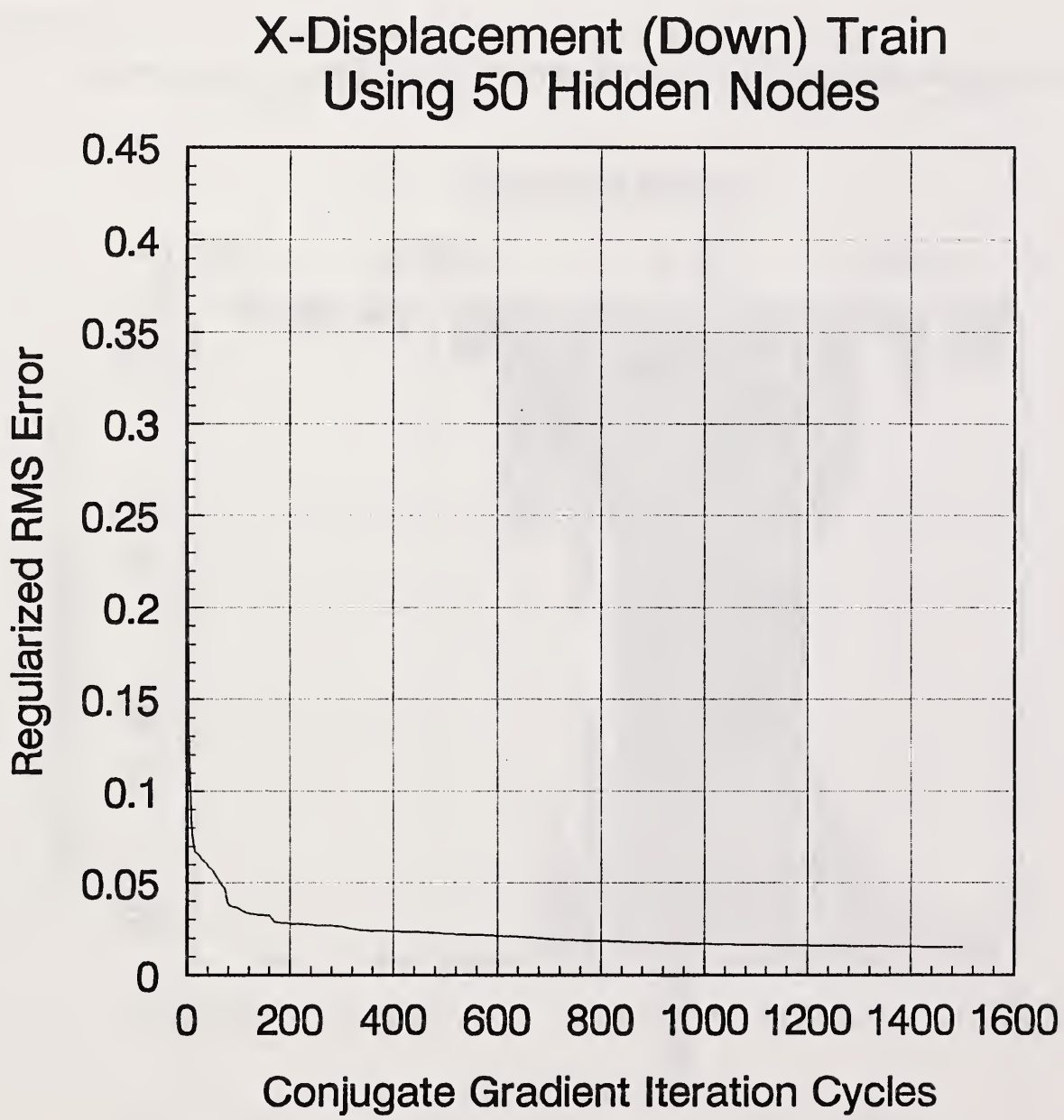

Figure 22. Regularized RMS error curve for training X-displacement downwards data aginst all active thermocouples using 50 hidden nodes. Regularized RMS errors are in millimeters.

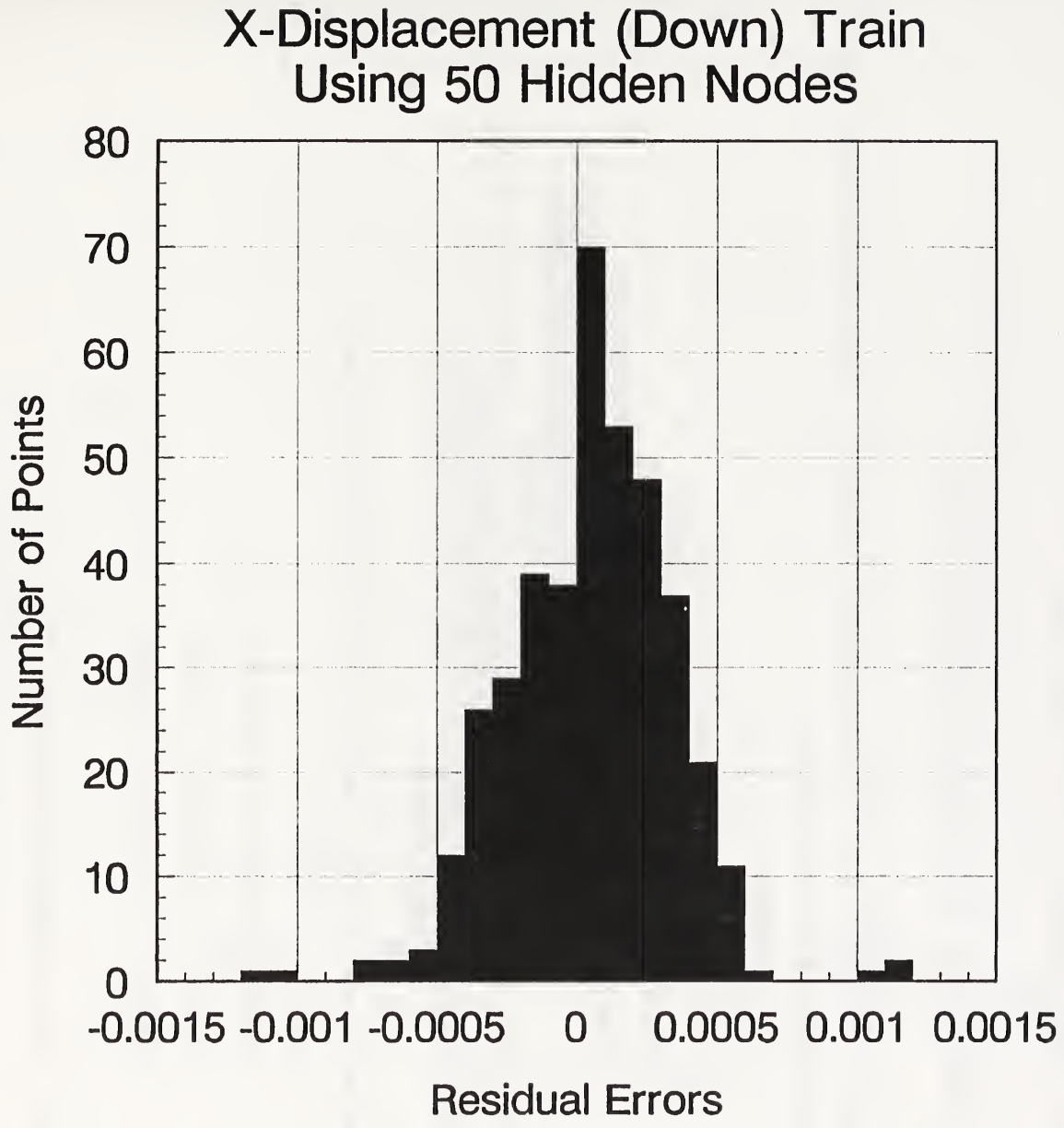

Figure 23 Histogram of residual errors for training X-displacement downwards data against all active thermocouples using 50 hidden nodes. Residual errors are in millimeters.

A test was performed on 198 data patterns. The statistics for the residuals were:

Mean = -0.3120217 E - 4 mm

Standard Deviation = 0.2221139 E - 3 mm

Minimum = -0.1341797 E - 2 mm

Maximum = 0.1040228 E - 2 mm

A histogram of the residuals is given in Figure 23A.

9.8 X-Displacement (Down) with 40 Hidden Nodes

This numerical experiment uses 36 thermocouples but reduces the number of hidden nodes used to 40. The fitting results reported were:

Mean = 0.8227056 E - 6 mm

Standard Deviation = 0.2425290 E - 3 mm

Minimum = -0.8013640 E - 3 mm

Maximum = 0.1270486 E - 2 mm

Figure 24 shows the regularized RMS error and Figure 25 shows the distribution of residual errors. In this case, there appears to be a general broadening of the distribution.

A test was performed on 198 data patterns and the residual statistics were:

Mean = -0.2879812 E - 4 mm

Standard Deviation = 0.2538050 E - 3 mm

Minimum = -0.1000706 E - 2 mm

Maximum = 0.1212418 E - 2 mm

A histogram of the residuals is given in Figure 25A.

10. Conclusions

The first observation that can be made is that the scaled conjugate gradient algorithm is a very consistent optimization technique for determining weights in a neural network. Table 3 assigns index numbers from 1 to 9 to the sample training runs made. In Table 4 the training run

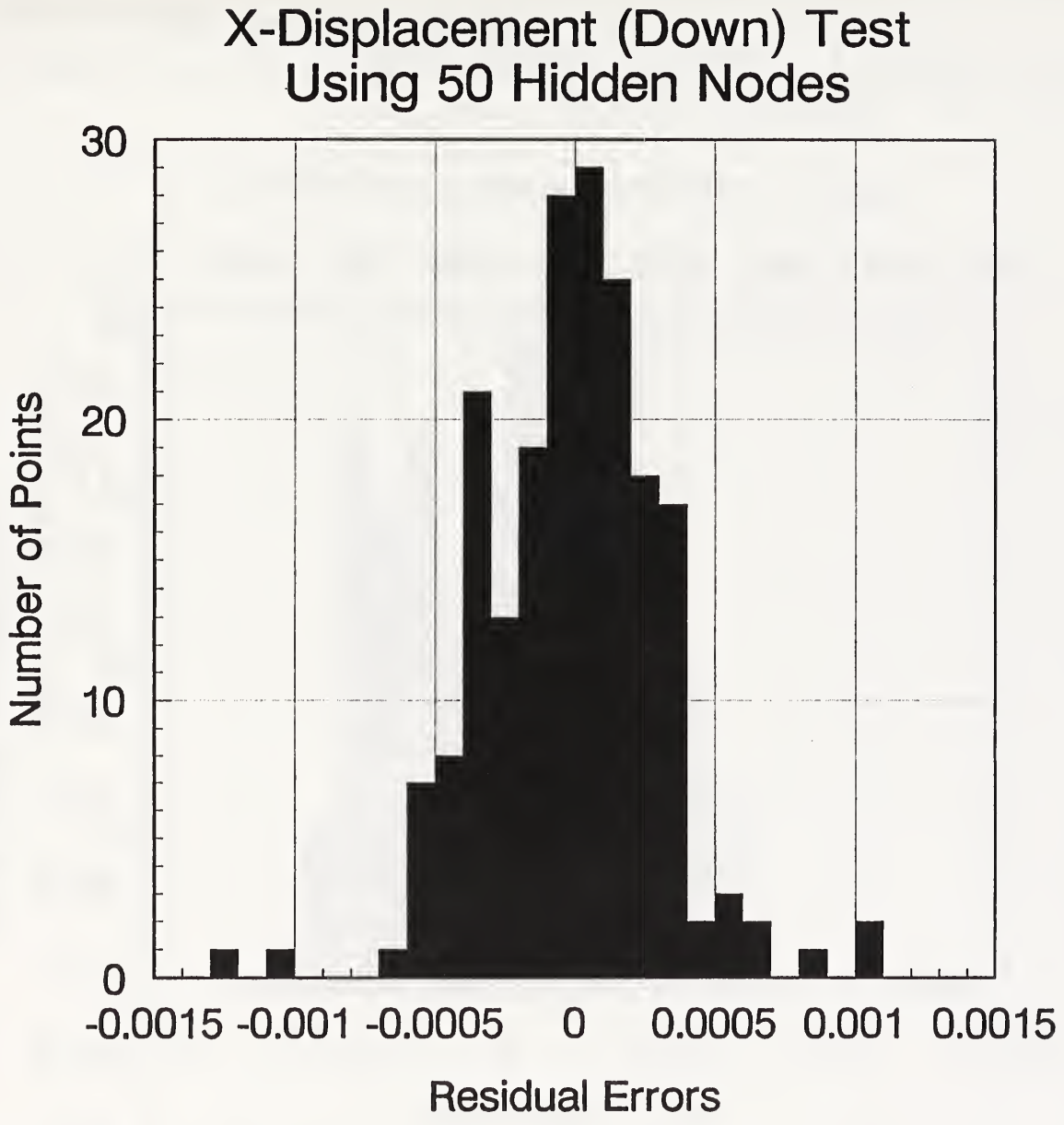

Figure 23A. Histogram of residual errors for testing X-displacement downwards data against all active thermocouples using 50 hidden nodes. Residual errors are in millimeters.

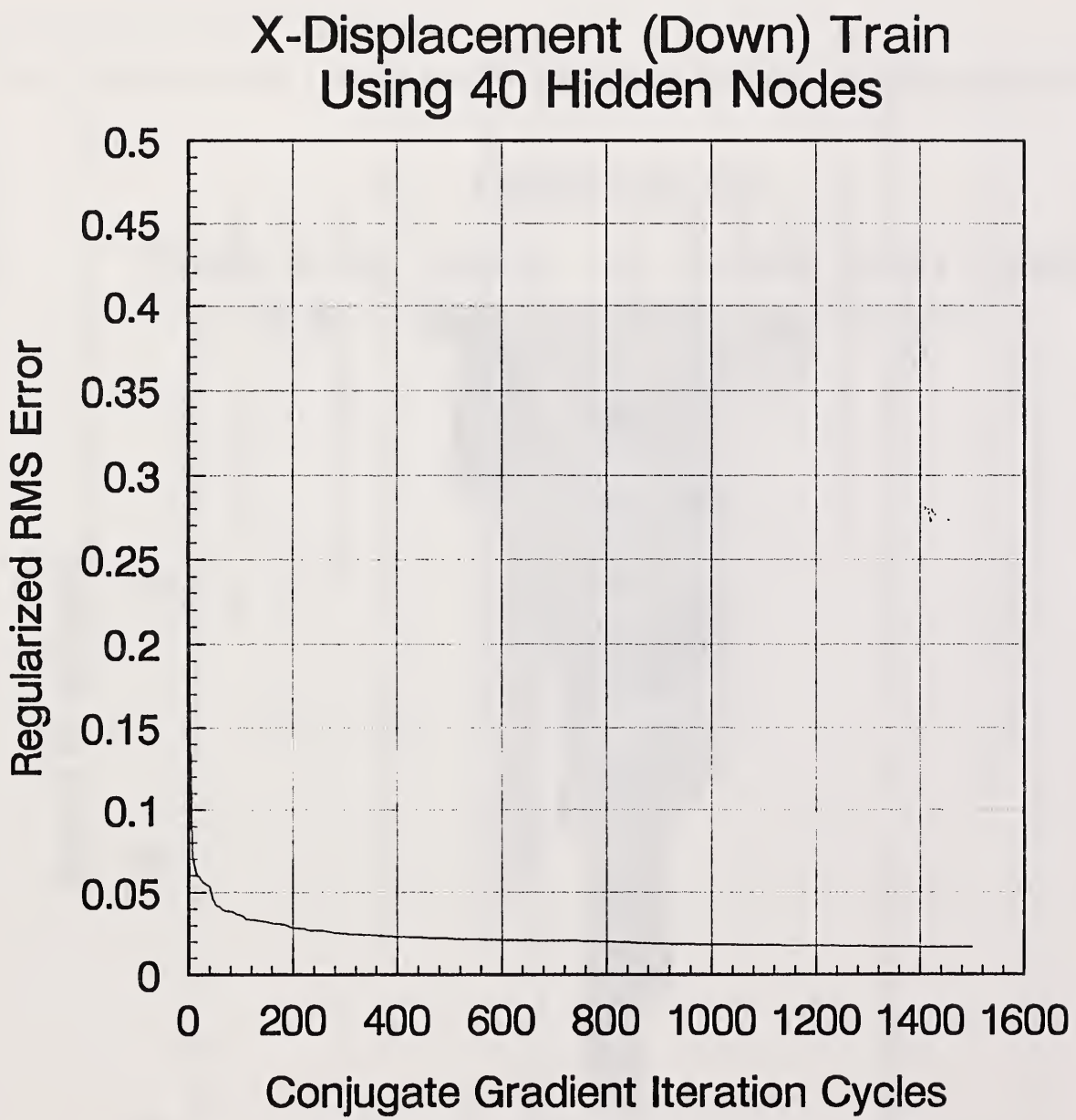

Figure 24. Regularized RMS error curve for training X-displacement downwards data against all active thermocouples using 40 hidden nodes. Regularized RMS errors are in millimeters.

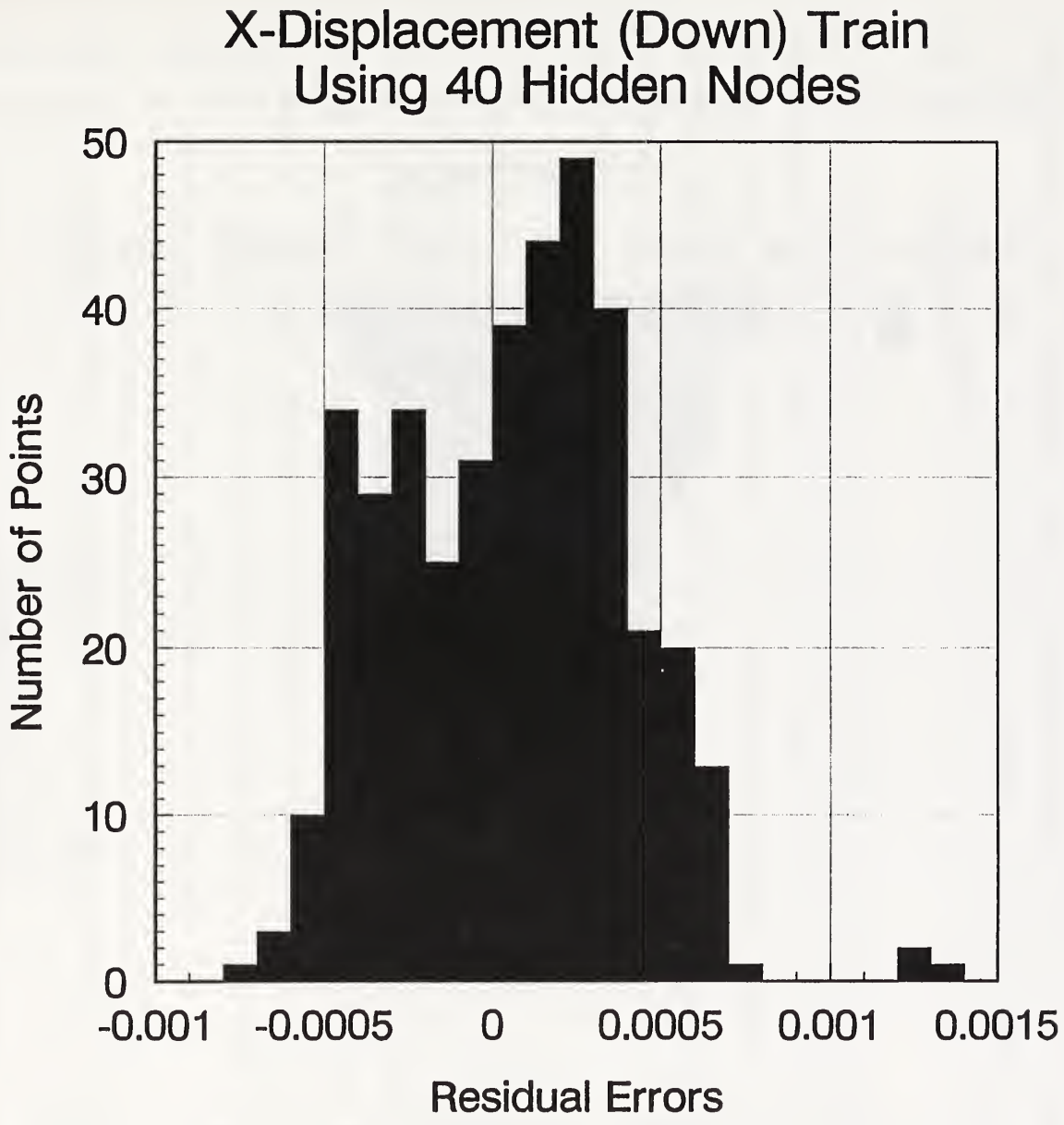

Figure 25. Histogram of residuals for training X-displacement downwards data against all active thermocouples using 40 hidden nodes. Residual errors are in millimeters

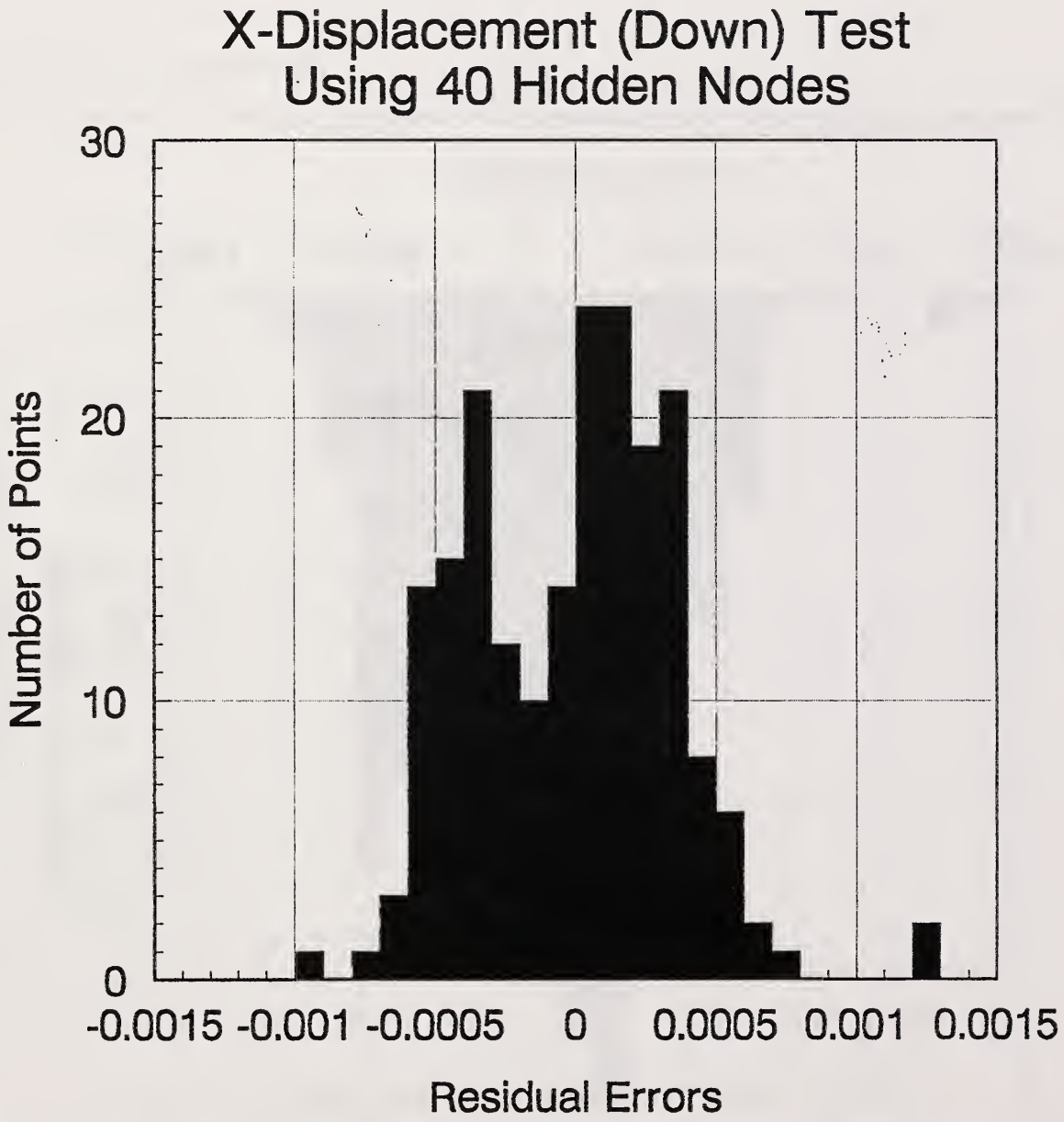

Figure 25A. Histogram of residuals for testing X-displacement downwards data against all active thermocouples using 40 hidden nodes. Residual errors are in millimeters.

statistics are summarized. The residual error statistics are all grouped very tightly as shown in Table 4. Since the number of connections in a network influences, the elasped time for a solution, the iteration time per connection was computed for each run. This only varied, with rounding, between 0.001 and 0.002 seconds. The elapsed time per connection varied between 1.3 and 2.3 seconds. All of the neural network computation were performed in FORTRAN on a CONVEX C3820 with the UNIX operating system. The other run time statistics are also tightly grouped. Table 4 gives the mean cpu time per iteration, the standard deviation of the cpu time for iteration, the total elasped time for each run, the iteration time per network connection and the elasped cpu time per network connection.

The next observation that can be made about the scaled conjugate gradient algorithm is that it is easy to use and converges rapidly as shown in the figures in section 9. It does not require multiple submission of randomized data patterns as the steepest descent algorithm sometimes requires. It adaptively adjusts its steps and does not require the user to guess at learning and momentum parameters as many implementations of the back-propagation algorithm requires. It can handle large network problems as consistently as moderate to small network problems.

Finally, when neural network methods are compared with regression methods for mapping, both have advantages and disadvantages. Regression analysis has the advantage that it is possible to identify a physical interpretation to the final fitted equations. It's disadvantage is the difficulty of selecting, for example, the polynomial forms of temperatures and nominal positions to enter the regression. Neural networks clearly will map thermal and nominal position data to position errors comparatively rapidly and consistently. However, it is difficult to assign a physical interpretation to the weighted network other than to say that it maps the training data.

As long as the objective remains only mapping data, neural networks demonstrate a clear advantage of ease-of-use. But, if real time control is required, then judicious choice of regression polynomials hold an advantage unless computer hardware can be built to evaluate neural networks with comparable speeds.

Run	Direction	Hidden Node	Thermocouple Selection
1	Down	60	All
2	Down	60	Operating
3	Up	60	Operating
3	Down	60	X-Axis Related
5	Down	60	Z-Axis Related
6	Up	60	X-Axis Related
7	Up	60	Z-Axis Related
8	Down	50	Operating
9	Down	40	Operating

Table 3. Run Summary

Run	1	2	3	4	5	6	7	8	9
Run Characteristics									
Direction	Down	Down	Up	Down	Down	Up	Up	Down	Down
Input Patterns	397	397	298	397	397	298	298	397	397
Thermocouples	40	36	36	9	14	7	12	36	36
Input nodes	41	37	37	9	15	8	13	37	37
Hidden nodes	60	60	60	60	60	60	60	50	40
Output nodes	1	1	1	1	1	1	1	1	1
Connections	2581	2341	2281	661	1021	601	901	1951	1561
Iteractions	1500	1500	1500	1500	1500	1500	1500	1500	1500
Residual Error Statistics (mm)									
Mean	0.33E-5	0.25E-5	0.73E-6	0.19E-5	0.99E-6	0.60E-5	0.19E-6	0.87E-6	0.83E-6
Std. Dev.	0.17E-3	0.17E-3	0.17E-3	0.3[illegible]E-3	0.19E-3	0.30E-3	0.33E-3	0.20E-3	0.24E-3
Minimum	-0.67E-3	-0.67E-3	-0.85E-3	-0.26E-2	-0.11e-2	-0.14E-2	-0.21E-2	-0.12E-2	-0.80E-3
Maximum	0.93E-3	0.76E-3	0.59E-3	0.16E-2	0.68E-3	0.12e-2	0.13e-2	0.11e-2	0.13e-2
Run Timing Statistics (Seconds)									
Mean Time/Iteration	3.01	2.69	1.99	1.01	1.37	0.71	0.94	2.25	1.81
Std. Dev.	0.28	0.21	0.11	0.59E-1	0.27E-1	0.53E-1	0.88E-1	0.12	0.21
Elasped Time	4511.6	4037.6	2979.1	1521.6	2053.0	1067.1	1405.9	3370.9	2722.2
Iteration Time/Connection	.0012	.0011	.0009	.001[illegible]	.0013	.0012	.0010	.0012	.0012
Elasped Time/Connection	1.75	1.72	1.31	2.30	2.01	1.78	1.56	1.7[illegible]	1.74

Table 4. Summary of training run results.

11. Bibliography

1. Donmez, M.A., Liu, C.R. and Barash, M.M., "A Generalized Mathematical Model for Machine Tool Errors," **Modeling, Sensing, and Control of Manufacturing Processes**, Bk. No. H00370, Srinivason, K., Hardt, D.L.E. and Komanduri, R., eds, ASME, New York , 1987.

2. Donmez, M.A., Blomquist, D.S., Hocken, R.J., Liu, C.R. and Barash, M.M., "A General Methodology for Machine Tool Accuracy Enhancement by Error Compensation," Precision Engineering, Vol. 4, 1986.

3. Donmez, M.A., Lee, K., Liu, C.R., Barash, M.M., "A Real-Time Error Compensation System for a Computerized Numerical Control Turning Center," Proceedings of the International Conference on Robotics and Automation, San Francisco, CA, April 7-10, 1986.

4. Wu, S. M., Anderson, R., "Survey and Analysis of Thermal Error Sensing and Compensation Techniques on Machine Tools," Dept. Mechanical Eng. and Appl. Mech., The University of Michigan, March, 1992.

5. Donmez, M.A., "A General Methodology for Machine Tool Accuracy Enhancement, Theory, Application and Implementation," Ph.D. Thesis, Purdue University, August, 1985.

6. Ritter, H., Martinetz, T., Schulten,K., **Neural Computation and Self-Organizing Maps,** Addison-Wesley, Reading, Massachusetts, 1992.

7. Cybenko, G., "Mathematical Problems in Neural Computing," **CSRD Rpt. No. 905,** Center for Supercomputing Research and Development, Univ. of Illinois, August, 1989.

8. Grother, P.J., Blue, J.L., "Training Feed Forward Neural Networks Using Conjugate Gradients," **NISTIR 4776,** National Institute of Standards and Technology, Gaithersburg, MD, February, 1992.

9. Møller, M.F., "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning," **Neural Networks,** Vol. 6, No. 4, 1993.

10. Zhang, G., Veale, R., Charlton, T., Borchardt, B., Hocken, R., "Error Compensation of Coordinate Measuring Machines," **Annals of the CIRP,** Vol. 34, No. 1, 1985.

11. Gilsinn, D.E., Bandy, H.T., Donmez, M.A., Greenspan,L., Harper, K., Wilkin, N., "Application of a Generic Machine Tool Error Model to a Turning Center," **NISTIR,** (To appear), National Institute of Standards and Technology, Gaithersburg, MD, 1993.

12. Girosi, F., Poggio, T., "Networks for Learning," in **Neural Networks, Concepts, Applications, and Implementations, Volume I,** Antognetti, P. and Milutinovic, V., eds., Prentice Hall, Englewood Cliffs, N.J., 1991.

13. Cybenko, G., "Approximation by Superposition of a Sigmoidal Functions," **Math. Control Signals Systems,** Vol. 2, 1989.

14. Rumelhart, D.E., McClelland, J.L., et al., **Parallel Distributed Processing, Exploration in the Microstructure of Cognition, Vol. 1: Foundations,** MIT Press, Cambridge, Massachusetts, 1986.

15. Hestenes, M.R., **Conjugate Direction, Methods in Optimization,** Springer-Verlag, New York, 1980.

16. Fletcher, R., **Practical Methods of Optimization,** John Wiley & Sons, Chichester, 1987.

17. Tikhonov, A.N., Arsenin, V.Y., **Solutions of Ill-Posed Problems,** John Wiley & Sons, New York, 1977.

18. Saarinen, S., Bramley, R., Cybenko, G., "Ill-Conditioning in Neural Network Training Problems," **SIAM Journ. on Sci. and Stat. Computing,** to appear 1993.

19. Filliben, J.J., "DATAPLOT - Introduction and Overview", **NBS SPEC. PUB. 667,** National Bureau of Standards, Gaithersburg, MD, June, 1984.

20. Hocken, R. J., "Quasistatic machine tool errors," Technology of Machine Tools, MTTF, 1980, 5.

21. Donmez, M. A., Yee, K. W., and Damazo, B., "Some Guidelines for Implementing Error Compensation on Machine Tools," NISTIR 5236, National Institute of Standards and Technology, Gaithersburg, MD (1993).

APPENDIX A

MAIN PROGRAM LISTING

```
c*****************************************************************
c
c                            NETCG.F
c
c*****************************************************************
c
c                         Main Routine
c
c This program implements a fully connected feedforward neural net.
c The program allows two alternatives:
c       1. If previous weights have been computed for the network
c          links then the program can be used to compute net
c          outputs from specified inputs.
c       2. If weights are not available then the program can be
c          used to compute weights by a least squares minimization
c          technique using predefined data patterns and a scaled
c          conjugate gradient technique.
c
c The scaled conjugate gradient technique employed is due to:
c M. F. Moller, "A scaled conjugate gradient algorithm for fast
c supervised learning," Neural Networks (To appear)
c
c Original Program by:
c       James L. Blue
c       Computing and Applied Mathematics Laboratory
c       National Institute of Standards and Technology
c       Gaithersburg, MD  20899
c
c Modified by:
c       David E. Gilsinn
c       Manufacturing Engineering Laboratory
c       National Institute of Standards and Technology
c       Gaithersburg, MD  20899
c
c Modifications made to apply neural nets as a mapping technique
c for machine tool errors as functions of nominal axis tool position
c and thermocouple readings. These mapped errors are incorporated
c into a geometric-thermal error correction model to predict
c machine tool errors in the work volume as a function of nominal
c tool position and thermocouple readings
c
c Run specification file: netcg.in
c This file must exist before program execution. It contains run
c information such as data file names, network parameters and
c convergence criteria.
c
c
c
c Definitions:
c       maxpat          - maximum allowed input patterns for teaching
c       maxins          - maximum number of input nodes allowed
c       maxhid          - maximum number of hidden nodes allowed
c       maxout          - maximum number of of output nodes allowed
c       maxwsize        - maximum number of weights allowed
c       w(maxwsize)     - array of weights. The first num1 are the
c                         weights on the input to hidden links and
c                         the rest are the weights on the hidden to
c                         output links.
c       wsav(maxwsize)  - temporary array for weights.
c       error           - current error function value
```

```
c       j               - output layer node index
c       p               - pattern number index
c       r               - run number index
c       nruns           - number of runs
c       nseed           - integer seed number for uniform random
c                         number generator
c       egoal           - convergence goal for the RMS of error
c       gwgoal          - convergence goal for the ratio of the
c                         RMS of the gradient to the RMS of the
c                         weights. Measure relative change of the
c                         error to the weights.
c       errdel          - required error reduction factor for the
c                         RMS of error every nfreq iterations
c       nfreq           - frequency for error checking and printing
c                         of progress.
c       npats           - number of training patterns for a given run
c       ninp            - number of input nodes
c       nhid            - number of hidden nodes
c       nout            - number of output nodes
c       ncalls          - number of calls to network evaluation
c                         subroutine forward
c       num1            - number of weights in links between input
c                         and hidden nodes
c       num2            - number of weights in links between hidden
c                         and output nodes
c       numw            - total number of weights = num1 + num2
c       vinp(maxpat,maxins+1)   - input data. Extra location
c                                 accounts for threshold dummy data.
c       vout(maxpat,maxout)     - computed output data.
c       target(maxpat,maxout)   - output data for training.
c       fpspec          - file id for file names and run parameters
c       fppat           - file id for training pattern data in fnpat
c       fpgetw          - file id for existing weights, if any, in
c                         fngetw
c       fpputw          - file id for storing final weights after
c                         training in fnputw
c       fprun           - file id for run summary and error messages
c                         in fnrun
c       wfactor         - regularization parameter
c
c
c*****************************************************************
c
c                       Parameter Specifications
c
c               MAXP - Maximum number of data patterns
c               MAXI - Maximum number of input units
c               MAXH - Maximum number of hidden units
c               MAXO - Maximum number of output units
c
c*****************************************************************
        parameter (MAXP = 3500)
        parameter (MAXI = 42)
        parameter (MAXH = 100)
        parameter (MAXO = 26)
        parameter (MAXW = MAXH*(MAXI+1) + MAXO*(MAXH+1))
c*****************************************************************
c
c                       Variable Declarations
c
```

```
c*****************************************************************
        real w (MAXW)
        real wsav(MAXW)
        real vinp(MAXP, MAXI+1), data(MAXP,MAXI+1)
        real vout(MAXP, MAXO), err(MAXP,MAXO), temp(MAXP)
        real target(MAXP, MAXO)
        real gm(MAXW), wnew(MAXW), pvect(MAXW),rvect(MAXW)
        real svect(MAXW), vhid(MAXP,MAXH+1),delta1(MAXP,MAXH)
        real delta2(MAXP,MAXO),hderiv(MAXH),oderiv(MAXO)
        integer itherm(MAXI), idum(MAXP), idatfl(MAXI+1)
        integer p
        integer r
        integer fpspec, fppat, fpgetw, fpputw, fprun, fpscr, fpgrph
        integer fpgrw1,fpgrw2, fpgrer
        character*80 title
        character*40 fnrun
        character*40 fnpat
        character*40 fngetw
        character*40 fnputw
        character*40 fngrph
        character*40 fngrw1, fngrw2, fngrer
        character*1  key, aster(80)
c*****************************************************************
c
c                       Begin Executables
c
c*****************************************************************
c
        do 2 i = 1,80
           aster(i) = '*'
 2         continue
        maxpat = MAXP
        maxins = MAXI
        maxhid = MAXH
        maxout = MAXO
        maxwsize = MAXW
c set file designation unit numbers
c
        fpspec = 9
        fppat = 10
        fpgetw = 11
        fpputw = 12
        fprun = 13
        fpscr = 14
        fpgrph = 15
        fpgrw1 = 16
        fpgrw2 = 17
        fpgrer = 18
c
c open file for run specifications
c
        open(fpspec, file = 'netcg.in', status = 'old')
        write(*,9000)
 9000   format(' Reading run specification file')
c
c>>>> Read first line: number of runs
c
        read(fpspec, *) nruns
        write(fprun,999) nruns
c
```

```
c Begin major loop over the number of runs.
c
       do 23000 r = 1, nruns
c
c*****************************************************************
c
c For each run read the run specification file
c
c*****************************************************************
c
c get the run title (<= 80 char)
c
        read(fpspec,*) title
c>>>> read file names, one name per line
c
c first file name: run output file name
c
        read(fpspec, *) fnrun
c
c second file name: pattern input file name
c
        read(fpspec, *) fnpat
c
c third file name: initial weights file name
c
        read(fpspec, *) fngetw
c
c fourth file name: final weights output file name
c
        read(fpspec, *) fnputw
c
c fifth file name: graphics output file (Regularized error during opt)
c
        read(fpspec,*) fngrph
c
c sixth file name: input-to-hidden weights graphics output
c
        read(fpspec,*) fngrw1
c
c seventh file name: hidden-to-output weights graphics output
c
        read(fpspec,*) fngrw2
c
c eighth file name: error graphics file (after fit)
c
        read(fpspec,*) fngrer
c
c>>>> read network parameters for the run
c
c number of input patterns
c number of input nodes
c number of hidden nodes
c number of output nodes
c regularization coefficient of |w| in error function (ex. 1.e-3)
c nseed is 0 if reading weights, else random number seed (ex. 12345)
c
        read(fpspec, *)npats,ninp,nhid,nout,wfactor,nseed
c
c test parameters read against bounds
c
```

```
         if(npats  .gt.  maxpat) then
            print *, 'Have ', npats, ' patterns; limit is ', maxpat
            stop
         else if(ninp  .gt.  maxins) then
            print *, 'Have ', ninp, ' input nodes; limit is ', maxins
            stop
         else if(nhid  .gt.  maxhid) then
            print *, 'Have ', nhid, ' hidden nodes; limit is ', maxhid
            stop
         else if(nout  .gt.  maxout) then
            print *, 'Have ', nout, ' output nodes; limit is ', maxout
            stop
         end if
c
c>>>> read convergence parameters for the current run
c
c number of iterations through the data for conjugate gradient routine
c       if set nonzero then the iterations are for training (ex. 200)
c       if zero then this flags a testing run
c goal for error (RMS) (ex. 0.01)
c goal for g (RMS) / w (RMS) (ex. 1.e-12)
c frequency for checking convergence (ex. 10)
c quit if error reduction too small (ex. 1.0)
c
         read(fpspec, *)niter,egoal,gwgoal,nfreq,errdel
c
c read data column flags. These determine which input columns
c enter the fitting process
c
         read(fpspec,*) (idatfl(i), i=1,20)
         read(fpspec,*) (idatfl(i), i = 21,41)
c
c>>>> end of run specification input
c
         close(fpspec)
c
c*****************************************************************
c
c end of run specification
c
c begin reading data patterns for training or testing
c
c*****************************************************************
c
c compute number of weights for each layer
c
c first get number of weights between input nodes and hidden nodes
c
         num1 = nhid*(ninp+1)
c
c next between the hidden nodes and the output nodes
c
         num2 = nout*(nhid+1)
c
c total number of weights both layers
c
         numw = num1 + num2
c
c open output file for the run
c
```

```
        open(fprun, file = fnrun, status = 'unknown')
c
c write output header
c
        write(fprun,9005)
 9005   format('              NEURAL NET RUN REPORT')
        write(fprun,9007) (aster(j),j=1,80)
 9007   format(1x,80a1)
        write(fprun,9008) title
 9008   format(1x,a80)
        write(fprun,9007) (aster(j),j=1,80)
        write(fprun,9009)
 9009   format(//)
        write(fprun,9019)
 9019   format('                    DATA SET')
        write(fprun,9009)
        write(fprun,9007) (aster(j),j=1,80)
        write(fprun,9009)
c
c check for training or testing run
c
        if(niter  .gt.  0) then
           write(fprun , 988) ' Training on ', fnpat
        else
           print *, ' '
           write(fprun , 988) ' Testing on ', fnpat
        end if
        write(fprun,9009)
c
c get the inputs and target patterns to be learned
c
        open(fppat, file = fnpat, status = 'old')
c
c next call reads data file to get network parmeters
c and sets up data arrays vinp and target
c
        write(*,9010)
 9010   format(' Reading pattern data')
        call getpat(fppat,data,idatfl,vinp,target,npats,ninp,nout,
     1              maxpat,maxins,maxout,itherm,idum,slope,tmax)
        close(fppat)
        write(fprun,9011) npats
 9011   format(' The number of data patterns is ',i6)
        write(fprun,9009)
        write(fprun,9007) (aster(j),j=1,80)
        write(fprun,9009)
c
c write out network parameters
c
        write(fprun,9021)
 9021   format(29x,'NETWORK SPECIFICATION')
        write(fprun,9009)
        write(fprun,9007) (aster(j),j=1,80)
        write(fprun,9009)
        write(fprun,9022)
 9022   format(12x,'INPUT NODES',11X,'HIDDEN NODES',11X,'OUTPUT NODES')
        write(fprun,9023)
 9023   format(/)
        write(fprun,9024) ninp,nhid,nout
 9024   format(15x,i2,2(22x,i2))
```

```
         write(fprun,9023)
         write(fprun,9025)
 9025    format(8x,'WEIGHTS (I TO H)',8x,'WEIGHTS (H TO O)',9x,'TOTAL WEIGHTS')
         write(fprun,9023)
         write(fprun,9026) num1,num2,numw
 9026    format(14x,i4,20x,i4,19x,i4)
         write(fprun,9009)
         write(fprun,9007) (aster(j),j=1,80)
         write(fprun,9009)
c
c*****************************************************************
c
c end of data pattern input
c
c begin entering initial network weights
c     either random
c     or previously saved
c
c*****************************************************************
c
c get initial weights
c
c if nseed is a positive integer generate random initial weights
c
         write(*,9020)
 9020    format(' Generating network weights')
         write(fprun,9027)
 9027    format(33x,'RUN PARAMETERS')
         write(fprun,9009)
         write(fprun,9007) (aster(j),j=1,80)
         write(fprun,9009)
         if(nseed  .le.  0) then
c
c otherwise get weights from the file fngetw
c
            open(fpgetw, file = fngetw, status = 'old')
            write(fprun, 987) fngetw
            call setwts(fpgetw, w, w(num1+1), 0,ninp,nhid,nout)
            close(fpgetw)
c
c generate random weights for nseed > 0
c
         else
            write(fprun, 986) nseed
            call setwts(0, w, w(num1+1), nseed,ninp,nhid,nout)
         end if
c
c*****************************************************************
c
c end of initial weight generation
c
c begin weight optimization by the conjugate gradient method
c
c*****************************************************************
c
c write out the regularization parameter
c
         write(fprun, 984) wfactor
c
c save the current weights for computing the rms change in weights
```

```
c for each training run
c
         do 23014 n = 1, numw
            wsav(n) = w(n)
23014       continue
c
c write out convergence criteria for a training run, ignore
c for a testing run
c
         if(niter  .gt.  0) then
            write(fprun,9028)
 9028       format('1')
            write(fprun,9007) (aster(j),j=1,80)
            write(fprun,9009)
            write(fprun,90281)
90281       format(28x,'RUN TERMINATION CRITERIA')
            write(fprun,9009)
            write(fprun,9007) (aster(j),j=1,80)
            write(fprun,9009)
            write(fprun,9029) niter
 9029       format(' Maximum number of iterations is ',i6)
            write(fprun,90282) nfreq
90282       format(' Error checking frequency is every ',i3,' iterations')
            write(fprun,90280)
90280       format(' Run is terminated on either max iterations or',/,
     &           ' one of the criteria below.')
            write(fprun,90283) egoal
90283       format(' RMS error <= ',g12.3)
            write(fprun,90284) gwgoal
90284       format(' (RMS of g) <= ',g12.3,' * (RMS of w)')
            write(fprun,90285) errdel,nfreq
90285       format(' (RMS err) > ',g12.3,' * (RMS err ',i4,
     &           ' iterations ago)')
         end if
c
c Open scratch and graphics files
c
         open(fpscr,status='scratch')
         open(fpgrph,file=fngrph,status='unknown')
         open(fpgrer,file=fngrer,status='unknown')
c
c initialize calls to subroutine forward. This subroutine evaluates
c the network output for a given input pattern given the current
c weights
c
        ncalls = 0
c
c Do the training or testing depending on niter
c       niter = 0 for testing
c       niter > 0 for training
c
c In either case the error is returned with an appropriate flag
c
         iwrt = 0
         write(*,9030)
 9030    format(' Entering optimization phase.')
          call optwts(niter, numw, w, error, gw, iter, ierr,num1,
     1               num2,wfactor,vinp,vout,target,npats,ninp,
     2               nhid,nout,maxpat,maxins,maxhid,maxout,
     3               maxwsize,ncalls,gm,wnew,pvect,rvect,svect,vhid,
```

```
     4                deltal,delta2,hderiv,oderiv,fpscr,fpgrph,fprun,
     &                nfreq,errdel,egoal,gwgoal,iwrt)
          close(fpgrph)
          close(fpscr)
c
c*****************************************************************
c
c end of weight optimization
c
c write out run information
c
c*****************************************************************
c
          write(*,9040)
 9040     format(' Output phase.')
          iwrt = 1
          call func(.false.,numw,w,error,gw,num1,num2,wfactor,
     &              vinp,vout,target,npats,ninp,nhid,nout,
     &              maxpat,maxins,maxhid,maxout,ncalls,vhid,
     &              delta1,delta2,hderiv,oderiv,iwrt,fprun)
          write(fprun,9028)
          write(fprun,9007) (aster(j),j=1,80)
          write(fprun,9009)
          write(fprun,9041)
 9041     format(35x,'RUN RESULTS')
          write(fprun,9009)
          write(fprun,9007) (aster(j),j=1,80)
          write(fprun,9009)
          do 9045 np = 1,npats
             do 9043 j = 1,nout
                target(np,j) = tmax + (target(np,j) - 0.99)/slope
                vout(np,j) = tmax + (vout(np,j) - 0.99)/slope
                err(np,j) = (target(np,j)-vout(np,j))
 9043           continue
 9045           continue
          ermn = 0.0
          targmn = 0.0
          errmax = 1.e-20
          errmin = 1.e20
          do 9048 np = 1,npats
             do 9046 j = 1,nout
                ermn = ermn + err(np,j)
                targmn = targmn + target(np,j)
                if (err(np,j) .gt. errmax) errmax = err(np,j)
                if (err(np,j) .lt. errmin) errmin = err(np,j)
 9046           continue
 9048           continue
                ermn = ermn/float(np*j)
                targmn = targmn/float(np*j)
c for one output node only
          do 9100 np = 1,npats
             temp(np) = err(np,1)
 9100        continue
          std = 0.0
          ssmn = 0.0
          ssreg = 0.0
          do 90491 np = 1,npats
             do 9049 j = 1,nout
                std = std + (err(np,j)-ermn)**2
                ssmn = ssmn + (target(np,j) - targmn)**2
```

```
                ssreg = ssreg + (vout(np,j) - targmn)**2
 9049           continue
90491           continue
                std = std/float(np*j - 1)
                std = sqrt(std)
                rsq = ssreg/ssmn
          write(fprun,90490) rsq
90490     format(//,28x,'R-SQ of nonlinear fit = ',g15.7)
          write(fprun,90492)
90492     format(//,28x,'Pointwise  Error')
          write(fprun,9023)
          write(fprun,90493)
90493     format(15x,'Mean',27x,'Standard Deviation')
          write(fprun,90494) ermn,std
90494     format(9x,g15.7,25x,g15.7)
c normalize eror distribution and compute Kolmogorov-Smirnov statistic
          do 9105 np = 1,npats
             temp(np) = (temp(np) - ermn)/std
 9105        continue
c
c compute Kolmogorov-Smirnov statistic and significance
c
          call ksone(temp,npats,d,prob)
          write(fprun,90498)
90498     format(//,14x,'Minimum',32x,'Maximum')
          write(fprun,90494) errmin,errmax
          write(fprun,9023)
          write(fprun,90495)
90495     format(8x,'Data',7x,'Output',7x,'Desired',7x,'Predicted',
     &         7x,'Absolute')
          write(fprun,90496)
90496     format(8x,'Index',7x,'Node',8x,'Output',8x,'Output',10x,'Error')
          do 90505 np = 1,npats
             do 90503 j = 1,nout
                write(fprun,90500) np,j,target(np,j),vout(np,j),err(np,j)
90500           format(8x,i4,8x,i4,5x,g12.4,1x,g12.4,7x,g12.4)
90503           continue
c
c generate error histogram display file
c
                write(fpgrer,90504) err(np,1)
90504           format(g12.4)
90505           continue
c
c display k-s statistic
c
          write(fprun,9110) d,prob
 9110     format(' Kolmogorov-Smirnov distance statistic = ',g15.7,/,
     &          ' Kolmogorov-Smirnov significance value = ',g15.7)
          close(fpgrer)
c
c write out convergence information for training runs, ignore for
c testiing runs
c
          if(niter  .gt.  0) then
             call endopt(fprun, iter, ncalls, ierr, error, gw)
c
c compute rms change in the weights for training runs, ignore for
c testing runs
c
```

```
            do 23020 n = 1, numw
                 wsav(n) = wsav(n) - w(n)
23020          continue
            dif = snrm2(numw, wsav, 1) / sqrt(float(numw))
            write(fprun, 985) dif
c
c skip to here for testing runs
c
         end if
c
c*****************************************************************
c
c write out final weights for training runs, ignore for testing
c
c*****************************************************************
c
c print final weights for training runs
c
         if(niter  .gt.  0) then
            open(fpputw, file = fnputw, status = 'unknown')
            open(fpgrw1,file = fngrw1,status = 'unknown')
            open(fpgrw2,file = fngrw2,status = 'unknown')
            call putwts(fpputw,fpgrw1,fpgrw2,w, w(num1+1),ninp,nhid,nout)
            close(fpputw)
            close(fpgrw1)
            close(fpgrw2)
            write(fprun, 983) fnputw
         end if
         close(fprun)
c
c*****************************************************************
c
c end of the run loop. Go back for another run specification
c if any
c
c*****************************************************************
c
c loop on runs
c
23000    continue
        stop
c
c*****************************************************************
c
c formats
c
c*****************************************************************
c
983   format(' Weights written to file ', a40)
984   format(' Regularization factor wfactor = ', 1pe12.3)
985   format(' Rms change in weights', f6.3)
986   format(' Random initial weights, seed ', i10)
987   format(' Initial weights from file ', a41)
988   format(' '/a12, a41)
999   format(' Doing ', i3, ' run(s)')
        end
c*****************************************************************
c
c                              setwts
c
```

```
c*****************************************************************
      subroutine setwts(fp, w1, w2, nseed,ninp,nhid,nout)
c*****************************************************************
c
c       Set up the initial weights for the neural net
c              Test on the unit number fp.
c              If fp >  0, read from there.
c              If fp <= 0, use pseudo-random weights in the range
c                          (-scale,scale), but first initializing
c                          the generator to nseed.
c              w1 - weights between input and hidden nodes
c              w2 - weights between hidden and ouput nodes
c
c*****************************************************************
      parameter(scale = 0.5)
      real w1(nhid, ninp+1)
      real w2(nout, nhid+1)
      integer fp, h, i, j, minp, mhid, mout
c
c*****************************************************************
c
c test unit number fp for whether to read weights or generate
c random weights
c
c if fp > 0 then read weights
c
c*****************************************************************
c
      if(fp  .gt.  0) then
c
c*****************************************************************
c
c come to this section to read previously stored weights
c
c*****************************************************************
c
c first get number of input, hidden and output nodes associated with
c the weights
c
       read(fp, *) minp, mhid, mout
c
c test these against network parameter specs for this run
c
       if((ninp  .ne.  minp) .or. (nhid  .ne.  mhid)  .or. (nout
     &    .ne.  mout)) then
          if(ninp  .ne.  minp) then
             print *, ' Saved network has ', minp, ' inputs; using ',
     &       ninp
             if(nhid  .ne.  mhid) then
               print *, ' Saved network has ', mhid,
     &         ' hidden nodes; using ', nhid
                if(nout  .ne.  mout) then
                  print *, ' Saved network has ', mout,
     &            ' outdput nodes; using ', nout
                  stop
                end if
             end if
          end if
       end if
c
```

```
c if network parameters match then
c read input to hidden nodes weights
c
         do 23040 h = 1, nhid
            read(fp, *) (w1(h,i), i = 1, ninp+1)
23040       continue
c
c then read hidden to output nodes weights
c
         do 23042 j = 1, nout
            read(fp, *) (w2(j,h), h = 1, nhid+1)
23042       continue
c
c*****************************************************************
c
c return after weights have been read in
c
c otherwise
c
c if fp = 0 come here to generate random weights
c in the range (-scale,scale) where scale is set at the beginning
c of this subroutine as a parameter.
c
c*****************************************************************
c
      else
c
c initialize random numbers to given seed
c
         z = uni(nseed)
c
c set input to hidden nodes weights
c
         do 23044 h = 1, nhid
            do 23046 i = 1, ninp+1
                 w1(h,i) = 2.0 * scale *(uni(0) - 0.5)
23046          continue
23044       continue
c
c set hidden to output nodes weights
c
         do 23048 j = 1, nout
            do 23050 h = 1, nhid+1
                 w2(j,h) = 2.0 * scale *(uni(0) - 0.5)
23050          continue
23048       continue
c
c*****************************************************************
c
c return section
c
c*****************************************************************
c
      end if
      return
      end
c
c*****************************************************************
c
c                              getpat
```

```
c
c*********************************************************************
        subroutine getpat(fppat,data,idatfl,vinp,target,npats,ninp,nout,
     1                    maxpat,maxins,maxout,itherm,idum,slope,tmax)
c*********************************************************************
c
c  Read the input, the target for each pattern
c
c  This subroutine depends on the particular data set formats
c
c  The current data set includes axis position, machine tool error
c  at the position and 40 thermocouple readings
c
c*********************************************************************
        integer fppat
        real vinp(maxpat, maxins+1),data(maxpat,maxins+1)
        real target(maxpat, maxout)
        integer i, j, p, idatfl(maxins+1)
        character*10 posit,diff
        integer itherm(maxins), idum(maxpat)
c
c read number of data patterns in file, number of input variables,
c number of output variables and a scale factor for the output data.
c This scale factor is used to produce the correct units if necessary.
c This section also checks against specifications. If there are
c discrepancies then it reports and stops.
c
        read(fppat, *) mpats, minp, mout, scale
        if(mpats  .lt.  npats  .or.  ninp  .ne.  minp  .or.  nout
     &  .ne.  mout) then
           if(npats  .gt.  mpats) then
            print *, ' File has ', mpats, ' patterns; using ', npats
           end if
           if(ninp  .ne.  minp) then
            print *, ' File has ', minp, ' inputs; using ', ninp
           end if
           if(nout  .ne.  mout) then
            print *, ' File has ', mout, ' outdput nodes; using ', nout
           end if
         end if
c
c read header line
c
        read(fppat,*) posit,diff,(itherm(j),j=1,3)
c
c get first set of data, scale the nominal axis position
c and map the target output to the interval (0,1)
c
        tmin = 1.e30
        tmax = -1.e30
        do 80 p = 1,npats
        read(fppat,*) idum(p), data(p,1), target(p,1),
     1          (data(p,j),j=2,4)
        if(target(p,1) .le. tmin) tmin = target(p,1)
        if(target(p,1) .ge. tmax) tmax = target(p,1)
80      continue
c
c transform target data to lie between 0.01 and 0.99 in order to
c match sigmoid range
c
```

```
        slope = 0.98/(tmax-tmin)
        do 85 p = 1,npats
           target(p,1) = slope*(target(p,1) - tmax) + 0.99
85      continue
c
c get the next 6 groups of thermocouple data
c
        do 100 k = 1,6
        read(fppat,*) (itherm(j), j=4+(k-1)*6,9+(k-1)*6)
        do 90 p = 1,npats
        read(fppat,*) (data(p,j),j=5+(k-1)*6,10+(k-1)*6)
90      continue
100     continue
c
c get data for the 40-th thermocouple
c
        read(fppat,*) itherm(40)
        do 110 p = 1,npats
        read(fppat,*) data(p,41)
110     continue
c
c copying nonzero columns to vinp
c
        minp = 0
        do 182 j = 1,41
           if (idatfl(j) .ne. 0) then
              minp = minp+1
              do 181 p = 1,npats
                 vinp(p,minp) = data(p,j)
 181             continue
           endif
 182       continue
c
c the last weight of each hidden node is really its bias,
c so its input value is 1
c
        do 183 p = 1,npats
        vinp(p,minp+1) = 1.0
 183    continue
        ninp = minp
        return
        end
c*********************************************************************
c
c                                 func
c
c*********************************************************************
        subroutine func(dograd, numw, wt, err, gm,num1,num2,wfactor,
     1                  vinp,vout,target,npats,ninp,nhid,nout,
     2                  maxpat,maxins,maxhid,maxout,ncalls,vhid,
     3                  delta1,delta2,hderiv,oderiv,iwrt,fprun)
c*********************************************************************
c
c This subroutine regularizes the error and negative gradient returned
c from the neural network evaluation function forward
c
c*********************************************************************
        logical dograd
        real wt(*), err, gm(*)
        real vinp(maxpat,maxins+1),vout(maxpat,maxout)
```

```
      real target(maxpat,maxout), vhid(maxpat,maxhid+1)
      real delta1(maxpat,maxhid), delta2(maxpat,maxout)
      real hderiv(maxhid), oderiv(maxout)
      integer fprun
c
c Evaluate the neural net function
c
      call forward(dograd, wt, wt(num1+1), err, gm, gm(num1+1),vinp,
     1             vout,target,npats,ninp,nhid,nout,maxpat,maxins,
     2             maxhid,maxout,ncalls,vhid,delta1,delta2,hderiv,
     3             oderiv,iwrt,fprun)
c
c average the norm squared of the weight vector.
c
      wsq = sdot(numw, wt, 1, wt, 1) / (2 * numw)
c
c This is the Tychonov regularization step. wfactor is the
c regularization parameter
c
      err = err + wfactor * wsq
      wf = wfactor / numw
c
c return the regularized negative gradient if needed. Note the
c gm is already negative on return from forward. The "-" sign
c just propagates the negative to the second term.
c
      if(dograd .eq. .true.) then
       do 23105 n = 1, numw
          gm(n) = gm(n) - wf * wt(n)
23105     continue
      end if
      return
      end
c*********************************************************************
c
c                              forward
c
c*********************************************************************
      subroutine forward(dograd, w1, w2, error, gm1, gm2,vinp,
     1                   vout,target,npats,ninp,nhid,nout,maxpat,
     2                   maxins,maxhid,maxout,ncalls,vhid,delta1,
     3                   delta2,hderiv,oderiv,iwrt,fprun)
c*********************************************************************
c
c Calculate outputs and (optionally) negative gradient. This is the
c main neural net evaluation function. It presumes a fully connected
c feedforward neural net with two active layers, a hidden one and
c an output one. The unit activation functions are the sigmoid
c functions.
c
c*********************************************************************
      logical dograd
      real w1(nhid, ninp + 1)
      real w2(nout, nhid + 1)
      real gm1(nhid, ninp + 1)
      real gm2(nout, nhid + 1)
      real vinp(maxpat, maxins+1)
      real vout(maxpat, maxout)
      real target(maxpat, maxout)
      real vhid(maxpat, maxhid+1)
```

```
      real delta2(maxpat, maxout)
      real delta1(maxpat, maxhid)
      real hderiv(maxhid)
      real oderiv(maxout)
      integer h, i, j, p, fprun
c
c SMIN should be small enough so that exp(SMIN) is negligible,
c but large enough so that exp(-SMIN) does not overflow.
c
      parameter (smin = -40.0)
c
c accumulate calls to the neural net evaluator
c
      ncalls = ncalls +1
c
c initialize error fo accumulation
      error = 0.0
c
c The object of this loop over all patterns is to compute the sum
c of the output errors over all of the patterns and to optionally
c compute the network delta factors for each network layer as a
c function of each pattern.
c
      do 23107 p = 1, npats
       do 23109 h = 1, nhid
c
c Accumulate the input signals to each hidden layer node as the
c inner product of the vector of weights on all of the links
c from the input nodes and the vector of all of the inputs for the
c p-th pattern. sdot is the dot product function from the BLAS
c package and essentially performs
c               sum = 0
c               do i = 1, ninp+1
c                    sum = sum + w1(h,i) * vinp(p,i)
c
          sum = sdot(ninp+1, w1(h,1), nhid, vinp(p,1), maxpat)
c
c assume the exponential sigmoid function for the activation function
c and evaluate the output at each hidden node as a function of the
c input signal. Test for underflow first.
c
          if(sum  .ge.  smin) then
               vhid(p,h) = 1.0 / (1.0 + exp(-sum))
          else
               vhid(p,h) = 0.0
          end if
c
c compute the derivative of the hidden node output with respect to
c the total input signal. i.e. partial of vhid w. r. t. sum.
c
          hderiv(h) = vhid(p,h)*(1.0-vhid(p,h))
23109     continue
c
c the last weight of each output node is really its bias,
c so its output value is 1
c
      vhid(p,nhid+1) = 1.0
c
      do 23113 j = 1, nout
c
```

```
c Compute the input signal to each output node as the inner product of
c the weight vector between the hidden nodes and the output nodes
c and the output vector of values from the hidden nodes. sdot performs
c
c                       sum = 0
c                       do h = 1, nhid+1
c                             sum = sum + w2(j,h) * vhid(p,h)
c
            sum = sdot(nhid+1, w2(j,1), nout, vhid(p,1), maxpat)
c
c assume the exponential sigmoid function as the activation function
c at each output node. Test for underflow.
c
            if(sum  .ge.  smin) then
                 vout(p,j) = 1.0 / (1.0 + exp(-sum))
            else
                 vout(p,j) = 0.0
            end if
c
c compute the derivative of the output with respect to the total input
c signal. i.e. partial of vout w. r. t. sum.
c
            oderiv(j) = vout(p,j)*(1.0-vout(p,j))
c
c accumulate error function over the patterns. The division by 2 in
c the error function definition is done once below.
c
            error = error + (target(p,j) - vout(p,j)) ** 2
23113       continue
c
c if the negative gradient is optionally required generate the
c the network deltas.
c
         if(dograd .eq. .false.) goto 23117
c
c output deltas
c
            do 23119 j = 1, nout
                 delta2(p,j) = (target(p,j) - vout(p,j)) * oderiv(j)
23119          continue
c
c hidden deltas
c
            do 23121 h = 1, nhid
c
c perform an inner product here with sdot
c
c                       sum = 0.0
c                       do j = 1, nout
c                             sum = sum + delta2(p,j) * w2(j,h)
c
                 sum = sdot(nout, delta2(p,1), maxpat, w2(1,h), 1)
                 delta1(p,h) = sum * hderiv(h)
23121          continue
c
c this is the end of the loop over all patterns
c
23117    continue
23107    continue
c
```

```
c Average error per node over all patterns and nodes
c
        div = (npats * nout)
        error = error / (2 * div)
        if(dograd .eq. .false.) goto 23123
c
c optionally generate the negative gradients with respect to the
c link weights
c
c first calculate negative gradient of error
c with respect to input weights
c
         do 23125 h = 1, nhid
            do 23127 i = 1, ninp+1
                 gm1(h,i) = sdot(npats, delta1(1,h), 1, vinp(1,i), 1) /
     &           div
23127          continue
23125       continue
c
c finally with respect to output weights
c
         do 23129 j = 1, nout
            do 23131 h = 1, nhid+1
                 gm2(j,h) = sdot(npats, delta2(1,j), 1, vhid(1,h), 1) /
     &           div
23131          continue
23129       continue
23123 continue
        return
        end
c*****************************************************************
c
c                              putwts
c
c*****************************************************************
        subroutine putwts(fp,fpgrw1,fpgrw2, w1, w2,ninp,nhid,nout)
c*****************************************************************
c
c               Write the weights to file with unit number fp.
c               Weights for each node start on a new line.
c
c               Write weights to graphics file
c
c*****************************************************************
        integer fp,fpgrw1,fpgrw2
        real w1(nhid, ninp+1)
        real w2(nout, nhid+1)
        integer h, i, j
        write(fp, 998) ninp, nhid, nout
        do 23133 h = 1, nhid
         write(fp, 999) (w1(h,i), i = 1, ninp+1)
23133    continue
        do 23135 j = 1, nout
         write(fp, 999) (w2(j,h), h = 1, nhid+1)
23135    continue
         noutp1 = nout + 1
         nip1 = ninp + 1
         nip2 = ninp + 2
         nhp1 = nhid + 1
         izero = 0
```

```
         write(fpgrw1,9000) nip2,nhp1
 9000    format(i6,1x,i6)
         write(fpgrw1,9010) izero,(i,i=1,nhid)
 9010    format(i6,100i15)
         do 23200 i = 1,nip1
            write(fpgrw1,9020) i,(w1(h,i),h=1,nhid)
 9020       format(i6,100e15.6)
23200       continue
         write(fpgrw2,9000) nhp1,noutp1
         do 23250 h = 1,nhp1
            write(fpgrw2,9030) h,w2(1,h)
 9030       format(i6,e15.6)
23250       continue
       return
998   format(3i5)
999   format(1x,5e14.6)
       end
c*********************************************************************
c
c                              endopt
c
c*********************************************************************
       subroutine endopt(fprun, iter, ncalls, ierr, err, gw)
c*********************************************************************
c
c              Do some output at the end of the optimization
c
c*********************************************************************
      integer fprun
c
c if ierr is <= 0 signal error goal achieved
c
      assign 900 to ifmt
c
c if ierr = 1 then hit iteration limit without
c convergence within specified goals
c
      if(ierr  .eq.  1) then
         assign 901 to ifmt
c
c if ierr = 2 then run termination due to small gradient
c
      else if(ierr  .eq.  2) then
         assign 902 to ifmt
c
c if ierr = 3 then error return due to slow convergence
c
      else if(ierr  .eq.  3) then
         assign 903 to ifmt
      end if
c
      write(*, ifmt) iter, ierr
      write(fprun, ifmt) iter, ierr
      write(*, 989) iter, ncalls, err, gw
      write(fprun, 989) iter, ncalls, err, gw
c
      return
c
900   format(' Iter', i6, '; ierr ', i1, ' : achieved error goal')
901   format(' Iter', i6, '; ierr ', i1, ' : iteration limit')
```

```
902    format(' Iter', i6, '; ierr ', i1, ' : gradient small')
903    format(' Iter', i6, '; ierr ', i1,
     & ' : slow convergence of error')
989    format(' Used', i6, ' iterations; ', i6, ' function calls; Err ',
     & f6.3,'; |g|/|w| ', 1pe9.3)
       end
c*********************************************************************
c
c                                uni
c
c*********************************************************************
       real function uni(jd)
c*********************************************************************
c***revision date  810915
c***category no.  g4a23
c***keywords  random numbers, uniform random numbers
c***author    blue, james, scientific computing division, nbs
c             kahaner, david, scientific computing division, nbs
c             marsaglia, george, computer science dept., wash state univ
c
c***purpose  this routine generates quasi uniform random numbers on [0,1)
c             and can be used on any computer with at least 16 bit integers,
c             e.g., with a largest integer at least 32767.
c***description
c
c       this routine generates quasi uniform random numbers on the interval
c       [0,1).  it can be used with any computer with at least 16 bit
c       integers, e.g., with a largest integer at least equal to 32767.
c
c
c   use
c       first time....
c                   z = uni(jd)
c                     here jd is any  n o n - z e r o  integer.
c                     this causes initialization of the program
c                     and the first random number to be returned as z.
c       subsequent times...
c                   z = uni(0)
c                     causes the next random number to be returned as z.
c
c   machine dependencies...
c      mdig = a lower bound on the number of binary digits available
c             for representing integers.  this value is defaulted
c             internally to 16, but may be increased in line with
c             remark a below.
c
c   remarks...
c     a. this program can be used in two ways:
c        (1) to obtain repeatable results on different computers,
c            set 'mdig' to the smallest of its values on each, or,
c        (2) to allow the longest sequence of random numbers to be
c            generated without cycling (repeating) set 'mdig' to the
c            largest possible value.
c     b. the sequence of numbers generated depends on the initial
c          input 'jd' as well as the value of 'mdig'.
c          if mdig=16 (the default) one should find that
c            the first evaluation
c              z=uni(305) gives z=.027832881...
c            the second evaluation
c              z=uni(0) gives   z=.56102176...
```

```
c            the third evaluation
c              z=uni(0) gives   z=.41456343...
c            the thousandth evaluation
c              z=uni(0) gives   z=.19797357...
c
c***references  (none)
c***routines called  (none)
c***end prologue  uni
c
c**********************************************************************
c
      integer m(17)
c
      save i,j,m,m1,m2
c
c  note... if a fortran 77 compiler is  n o t  available, the preceding
c          save statement should be removed.
c
comment      data mdig / 16 /
      data mdig / 32 /
      data m(1),m(2),m(3),m(4),m(5),m(6),m(7),m(8),m(9),m(10),m(11),
     1    m(12),m(13),m(14),m(15),m(16),m(17)
     2                  / 30788,23052,2053,19346,10646,19427,23975,
     3                    19049,10949,19693,29746,26748,2796,23890,
     4                    29168,31924,16499 /
      data m1,m2,i,j / 32767,256,5,17 /
c***first executable statement  uni
      if(jd .eq. 0) go to 3
c  fill
      m1 = 2**(mdig-2) + (2**(mdig-2)-1)
      m2 = 2**(mdig/2)
      jseed = min0(iabs(jd),m1)
      if( mod(jseed,2).eq.0 ) jseed=jseed-1
      k0 =mod(9069,m2)
      k1 = 9069/m2
      j0 = mod(jseed,m2)
      j1 = jseed/m2
      do 2 i=1,17
        jseed = j0*k0
        j1 = mod(jseed/m2+j0*k1+j1*k0,m2/2)
        j0 = mod(jseed,m2)
    2   m(i) = j0+m2*j1
      i=5
      j=17
c  begin main loop here
    3 k=m(i)-m(j)
      if(k .lt. 0) k=k+m1
      m(j)=k
      i=i-1
      if(i .eq. 0) i=17
      j=j-1
      if(j .eq. 0) j=17
      uni=float(k)/float(m1)
      return
      end
c**********************************************************************
c
c                                 sdot
c
c**********************************************************************
```

```
      REAL FUNCTION SDOT(N,SX,INCX,SY,INCY)
C**********************************************************************
C
C To compute the inner product between vectors sx and sy with
C separate increments. This routine from the BLAS library.
C
C**********************************************************************
      REAL SX(*),SY(*)
C***FIRST EXECUTABLE STATEMENT  SDOT
      SDOT = 0.0E0
      IF(N.LE.0)RETURN
      IF(INCX.EQ.INCY) IF(INCX-1)5,20,60
    5 CONTINUE
C
C        CODE FOR UNEQUAL INCREMENTS OR NONPOSITIVE INCREMENTS.
C
      IX = 1
      IY = 1
      IF(INCX.LT.0)IX = (-N+1)*INCX + 1
      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
      SDOT = SDOT + SX(IX)*SY(IY)
      IX = IX + INCX
      IY = IY + INCY
   10 CONTINUE
      RETURN
C
C        CODE FOR BOTH INCREMENTS EQUAL TO 1
C
C
C        CLEAN-UP LOOP SO REMAINING VECTOR LENGTH IS A MULTIPLE OF 5.
C
   20 M = MOD(N,5)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
      SDOT = SDOT + SX(I)*SY(I)
   30 CONTINUE
      IF( N .LT. 5 ) RETURN
   40 MP1 = M + 1
      DO 50 I = MP1,N,5
      SDOT = SDOT + SX(I)*SY(I) + SX(I + 1)*SY(I + 1) +
     1   SX(I + 2)*SY(I + 2) + SX(I + 3)*SY(I + 3) + SX(I + 4)*SY(I + 4)
   50 CONTINUE
      RETURN
C
C        CODE FOR POSITIVE EQUAL INCREMENTS .NE.1.
C
   60 CONTINUE
      NS=N*INCX
      DO 70 I=1,NS,INCX
      SDOT = SDOT + SX(I)*SY(I)
   70 CONTINUE
      RETURN
      END
C**********************************************************************
C
C                                  snrm2
C
C**********************************************************************
      REAL FUNCTION SNRM2(N,SX,INCX)
```

```
c**********************************************************************
c
c This function computes the norm of the vector sx with increment
c allowed.  The routine is from the BLAS library.
c
c**********************************************************************
      INTEGER          NEXT
      REAL   SX(*),  CUTLO, CUTHI, HITEST, SUM, XMAX, ZERO, ONE
      DATA   ZERO, ONE /0.0E0, 1.0E0/
C
      DATA CUTLO, CUTHI / 4.441E-16,  1.304E19 /
C***FIRST EXECUTABLE STATEMENT  SNRM2
      IF(N .GT. 0) GO TO 10
       SNRM2  = ZERO
       GO TO 300
C
   10 ASSIGN 30 TO NEXT
      SUM = ZERO
      NN = N * INCX
C                                                 BEGIN MAIN LOOP
      I = 1
   20  GO TO NEXT,(30, 50, 70, 110)
   30 IF( ABS(SX(I)) .GT. CUTLO) GO TO 85
      ASSIGN 50 TO NEXT
      XMAX = ZERO
C
C                       PHASE 1.  SUM IS ZERO
C
   50 IF( SX(I) .EQ. ZERO) GO TO 200
      IF( ABS(SX(I)) .GT. CUTLO) GO TO 85
C
C                                PREPARE FOR PHASE 2.
      ASSIGN 70 TO NEXT
      GO TO 105
C
C                                PREPARE FOR PHASE 4.
C
  100 I = J
      ASSIGN 110 TO NEXT
      SUM = (SUM / SX(I)) / SX(I)
  105 XMAX = ABS(SX(I))
      GO TO 115
C
C                  PHASE 2.  SUM IS SMALL.
C                            SCALE TO AVOID DESTRUCTIVE UNDERFLOW.
C
   70 IF( ABS(SX(I)) .GT. CUTLO ) GO TO 75
C
C                   COMMON CODE FOR PHASES 2 AND 4.
C                   IN PHASE 4 SUM IS LARGE.  SCALE TO AVOID OVERFLOW.
C
  110 IF( ABS(SX(I)) .LE. XMAX ) GO TO 115
       SUM = ONE + SUM * (XMAX / SX(I))**2
       XMAX = ABS(SX(I))
       GO TO 200
C
  115 SUM = SUM + (SX(I)/XMAX)**2
      GO TO 200
C
C
```

```
C                  PREPARE FOR PHASE 3.
C
   75 SUM = (SUM * XMAX) * XMAX
C
C
C     FOR REAL OR D.P. SET HITEST = CUTHI/N
C     FOR COMPLEX      SET HITEST = CUTHI/(2*N)
C
   85 HITEST = CUTHI/FLOAT( N )
C
C                   PHASE 3.  SUM IS MID-RANGE.  NO SCALING.
C
      DO 95 J =I,NN,INCX
      IF(ABS(SX(J)) .GE. HITEST) GO TO 100
   95  SUM = SUM + SX(J)**2
      SNRM2 = SQRT( SUM )
      GO TO 300
C
  200 CONTINUE
      I = I + INCX
      IF ( I .LE. NN ) GO TO 20
C
C            END OF MAIN LOOP.
C
C            COMPUTE SQUARE ROOT AND ADJUST FOR SCALING.
C
      SNRM2 = XMAX * SQRT(SUM)
  300 CONTINUE
      RETURN
      END
c**********************************************************************
c
c                              optwts
c
c**********************************************************************
      subroutine optwts(itermax, num, w, rmserr, gw, iter, ierr,num1,
     1                  num2,wfactor,vinp,vout,target,npats,ninp,nhid,
     2                  nout,maxpat,maxins,maxhid,maxout,maxwsize,
     3                  ncalls,gm,wnew,p,r,s,vhid,delta1,delta2,
     4                  hderiv,oderiv,fpscr,fpgrph,fprun,nfreq,
     &                  errdel,egoal,gwgoal,iwrt)
c**********************************************************************
c
c     Solve neural net least squares problem by scaled conjugate gradients.
c     Return weight vector, error, |g|/|w|.
c     Stop if any of the following is true (return value as ierr)
c             0) rmserr <= egoal
c             1) Used itermax iterations.
c             2) Size of gradient vector < GWRATIO * size of weight vector
c             3) Error hasn't gone down by EFACTOR in nfreq iterations
c
c**********************************************************************
c the weights
      real w(num)
c
      real vinp(maxpat,maxins+1)
      real vout(maxpat,maxout)
      real target(maxpat,maxout)
      real vhid(maxpat,maxhid+1)
      real delta2(maxpat,maxout)
```

```
      real delta1(maxpat,maxhid)
      real hderiv(maxhid)
      real oderiv(maxout)
c negative gradients of error wrt w's
      real gm(maxwsize)
c new weights for temporary storage
      real wnew(maxwsize)
c direction vector
      real p(maxwsize)
c remembered negative gradient
      real r(maxwsize)
c second derivative info along p direction
      real s(maxwsize)
c number of steps since last restart
      integer icount
c number of consecutive failures
      integer fcount
      integer fpscr, fpgrph, fprun
      logical success
c
c starting value for xl (or lambda). This is lambda_1.
c
      parameter (xlstart = 0.01)
c
c*************************************************************
c
c       ******* INITIALIZATION SECTION ********
c
c*************************************************************
c
c test whether the maximum number of iterations is positive.
c If not exit with the computed function value (error) without the
c negative gradient. itermax is set to 0 when running tests.
c
      if (itermax .gt. 0) then
c
c
c
c When itermax > 0:
c Get initial function value and negative gradient
c Compute initial rms error and zero iteration counter
c Set counters and set initial search direction
c
c
c
         write(fprun, 999) itermax, num
 999     format(' max iterations = ',i6,' # of weights = ',i6)
c
c*****************************************************************
c
c ********** GET INITIAL FUNCTION AND NEGATIVE GRADIENT *********
c
c*****************************************************************
c
c get inital error and negative gradient
c
         call func(itermax  .gt.  0, num, w, error, gm,num1,num2,
     1           wfactor,vinp,vout,target,npats,ninp,nhid,nout,
     2           maxpat,maxins,maxhid,maxout,ncalls,vhid,
     3           delta1,delta2,hderiv,oderiv,iwrt,fprun)
```

```
c
c get the norm of the weight vector
c
          wsiz = snrm2(num, w, 1)
c
c initialize the iteration counter if itermax > 0
c
          iter = 0
c
c compute the rms error.
c
c Note rms error is the sqrt of the sums of square differences
c with no leading 1/2. error has a leading 1/2 that must be canceled
c by the 2 multiplier. Note that error includes division by
c the product of the number of patterns and the number of
c outputs when passed back from func. Therefore the next line forms a
c legitimate rms error by computing the norm of a vector and dividing
c by the square root of its length.
c
          rmserr = sqrt(error * 2)
c
c initialize error flag to indicate exit with iteration count
c larger than itermax. This is the default setting.
c
          ierr = 1
c
c initialize relative distance for numerical derivative
c
          sigma = 1.e-4
c
c initialize lambda to lambda_1 > 0
c
          xl = xlstart
c
c initailize lambda_1 bar to 0
c
          xlb = 0
c
c initialize positive definiteness parameter to 0
c
          deltak = 0
c
c initialize logical flag identifying that a reduction in
c error function can be made
c
          success = .true.
c
c save the current rms error
c
          rmsold = rmserr
c
c identify how often to restart the algorithm. The conjugate
c gradient algorithms must be restarted after the total number
c of iterations becomes a multiple of the number of unknowns.
c This is because the function being minimized is not necessarily
c quadratic.
c
          iover = num
c
c initialize the number of iterations since last restart. This
```

```
c counter gets reset after each restart. It is used to test
c against iover.
c
          icount = 0
c
c initialize the number of failed iterations in a row. An iteration
c is considered a failure if it does not reduce the error function.
c This counter will be used as a termination check. The subroutine
c will stop on too many failures in a row.
c
          fcount = 0
c
c initialize the number of iterations since last convergence check
c
          ncount = 0
c
c save the max iterations
c
          iter = itermax
c
c*********************************************************
c
c      ******** GET INITIAL SEARCH DIRECTION********
c
c*********************************************************
c
c initialize both the step direction and current steepest
c descent array to the initial steepest descent
c
          do 23006 n = 1, num
             p(n) = gm(n)
             r(n) = gm(n)
23006     continue
c
c*********************************************************
c
c     ******** INITIALIZE ITERATION COUNTER *********
c
c*********************************************************
c
c initialize the iteration counter. This index continues to
c accumulate no matter how many restarts.
c
          k = 0
          write(*,9000)
 9000     format(' Iteration, rmserr')
       else
c
c*********************************************************
c
c  ******** RUNNING TESTS ONLY *********
c
c*********************************************************
c
c When itermax <= 0:
c Get initial function value and rms error
c Set return flag
c
c
c
```

```
c get inital error only
c
          call func(itermax  .gt.  0, num, w, error, gm,num1,num2,
     1              wfactor,vinp,vout,target,npats,ninp,nhid,nout,
     2              maxpat,maxins,maxhid,maxout,ncalls,vhid,
     3              delta1,delta2,hderiv,oderiv,iwrt,fprun)
c
c get the norm of the weight vector
c
          wsiz = snrm2(num, w, 1)
c
c set error index if itermax <= 0 then exit with rms error
c
          iter = -1
c
c compute the rms error no matter what itermax is
c
          rmserr = sqrt(error * 2)
c
c return if itermax <= 0. This would happen on testing runs
c and not training runs
c
          return
        end if
c
c*************************************************************
c
c        ****** MAIN ITERATION LOOP *************
c
c*************************************************************
c
c Entry for the main iteration loop
c if the iteration counter hits the maximum iterations then exit the
c routine
c
23008   if(k  .ge.  itermax) then
          goto 23009
        end if
c
c increment the counter for the number of iterations since
c the last restart
c
        icount = icount + 1
c
c get the norm of the current direction vector
c
        psiz = snrm2(num, p, 1)
c
c square this norm
c
        psq = psiz**2
c
c if a reduction in the error can be made compute the
c quadratic terms otherwise only adjust parameters with current
c quadratic information by skipping to step 3 of the algorithm.
c A reduction in error cannot be made if the comparison
c delta is < 0. If so try scaling the current Hessian again, get
c a new step size and recompute the comparison parameter delta.
c
        if(success .eq. .true.) then
```

```
c
c*********************************************************
c
c******** GET SECOND ORDER DIRECTIONAL DERIVATIVE *********
c
c*********************************************************
c
c a reduction in the error function can be made
c get divisor for approximate second derivative info
c
            sigmak = sigma * wsiz / psiz
            do 23012 n = 1, num
                 wnew(n) = w(n) + sigmak * p(n)
23012          continue
c
c get error and negative gradient at the new weight vector point
c
            call func(.true., num, wnew, enew, s,num1,num2,wfactor,
     1                 vinp,vout,target,npats,ninp,nhid,nout,
     2                 maxpat,maxins,maxhid,maxout,ncalls,vhid,
     3                 delta1,delta2,hderiv,oderiv,iwrt,fprun)
c
c Compute the second order directional derivative along p
c
            do 23014 n = 1, num
                 s(n) = (gm(n) - s(n)) / sigmak
23014          continue
c
c compute definiteness check parameter
c
            deltak = sdot(num, s, 1, p, 1)
         end if
c
c*********************************************************
c
c  ********* MAKE HESSIAN POSITIVE DEFINITE ***********
c
c*********************************************************
c
c section to scale approximate Hessian
c and definiteness parameter
c
c skip to this section if a reduction
c in error could not be previously done.
c Attempt to scale and try stepping again.
c The subroutine will terminate if too many attempts
c in a row are made to try to reduce the error and
c fail.
c
c save lambdak - lambdak_bar
c
      c = xl - xlb
c
c if it is 0 skip the scaling
c
      if(c  .ne.  0.0) then
c
c scale the approximate Hessian and the definiteness parameter
c
         do 23018 n = 1, num
```

```
            s(n) = s(n) + c * p(n)
23018          continue
         deltak = deltak + c * psq
      end if
c
c
c
c test for the definiteness of the Hessian
c if it is not positive definite (deltak <=0) scale the
c second order term to make it positive definite
c before taking a new step.
c If it is positive definite skip to get a new
c step size
c
c
c
      if(deltak  .le.  0) then
         c = xl - 2 * deltak / psq
         do 23022 n = 1, num
            s(n) = s(n) + c * p(n)
23022          continue
         xlb = 2 * (xl - deltak / psq)
         deltak = - deltak + xl * psq
         xl = xlb
      end if
c
c*************************************************************
c
c    ******** COMPUTE NEW STEP LENGTH **********
c
c*************************************************************
c
      xmu = sdot(num, p, 1, r, 1)
      alpha = xmu / deltak
c
c*************************************************************
c
c   ********* COMPUTE NEW WEIGHT VECTOR ************
c
c*************************************************************
c
      do 23024 n = 1, num
         wnew(n) = w(n) + alpha * p(n)
23024       continue
c
c*************************************************************
c
c  ******** COMPUTE QUADRATIC TEST PARAMETER ********
c
c*************************************************************
c
c get new function value and negative gradient
c compute quadratic approximation test parameter
c A reduction in error can be made if delta >=0
c otherwise delta < 0 means a reduction could not
c be made.
c
      call func(.true., num, wnew, enew, gm,num1,num2,
     1             wfactor,vinp,vout,target,npats,ninp,
     2             nhid,nout,maxpat,maxins,maxhid,maxout,
```

```
     3             ncalls,vhid,delta1,delta2,hderiv,oderiv,iwrt,fprun)
      delta = 2 * deltak * (error - enew) / xmu**2
c
c if a reduction in error can be made (delta >= 0) get a new
c conjugate direction, otherwise get a count of the failures.
c
      if(delta  .ge.  0.0) then
c
c*************************************************************
c
c ******** SETUP FOR NEW CONJUGATE DIRECTION **********
c *** OTHERWISE TEST FOR TOO MANY UNSUCCESSFUL STEPS **
c
c*************************************************************
c
c a reduction in error was made
c set up new conjugate direction and
c adjust parameters to maintain a
c trustworthy quadratic approximation
c
c iteration counter can be updated as well as the count of the number
c of iterations since the last convergence check. Since this is a
c successful iteration the number of failed iterations in a row can
c be reset to 0
c
         k = k + 1
         write(fpscr,*) k,rmserr
         if (mod(k,nfreq) .eq. 0) write(*,*) k,rmserr
         ncount = ncount + 1
         fcount = 0
c
c*************************************************************
c
c ********** UPDATE THE WEIGHT VECTOR ************
c
c*************************************************************
c
         do 23028 n = 1, num
            w(n) = wnew(n)
23028          continue
c
c compute its norm
c
         wsiz = snrm2(num, w, 1)
c
c update the current error function value
c
         error = enew
         xlb = 0
         success = .true.
c
c*************************************************************
c
c ********** TEST FOR ALGORITHM RESTART ************
c
c*************************************************************
c
         if(mod(icount, iover)  .eq.  0)then
c
c*************************************************************
```

```
c
c ********** GET NEW CONJUGATE DIRECTION ************
c
c*********************************************************
c
c restart by setting current direction to steepest descent
c
            do 23032 n = 1, num
               p(n) = gm(n)
23032             continue
         else
c
c if there is no restart get a new conjugate direction
c
            beta = (sdot(num, gm, 1, gm, 1) - sdot(num, gm, 1, r, 1))
     &           / xmu
            do 23034 n = 1, num
               p(n) = gm(n) + beta * p(n)
23034             continue
         end if
c
c After getting new direction update current steepest descent
c
         do 23036 n = 1, num
            r(n) = gm(n)
23036       continue
c
c*************************************************************
c
c  ****** TEST QUADRATIC TRUSTWORTHINESS ***********
c
c*************************************************************
c
         if(delta  .ge.  0.75) then
c
c quadratic approximation is trustworthy at this point, can reduce
c lambda to increase step size
c
            xl = xl / 2
c
c test whether delta < 0.25
c
c not nearly as good a quadratic approximation therefore increase xl to
c reduce the step length
c
         else if(delta  .lt.  0.25) then
            xl = 4.0 * xl
         end if
      else
c
c*************************************************************
c
c ********** TEST FOR TOO MANY UNSUCCESSFUL STEPS ***********
c ************** COME HERE IF DELTA < 0 *********************
c
c*************************************************************
c
c end if delta < 0 too many times
c
c unsuccessful step and error cannot be reduced
```

```
c increment iteration failure counter
c
        xlb = xl
        success = .false.
        fcount = fcount + 1
c
c Do not exit on a single failure to reduce the error. Attempt to
c scale another time. However set two failures as a limit.
c If there are fewer than two failures in a row then go on to step 8
c to adjust lambdak upwards which reduces the step size and possibly
c the local error
c
        if(fcount  .gt.  2) then
c
c if the number of iterations since the last restart is not greater
c than the iteration failure counter then exit
c
            if(icount  .gt.  fcount) then
c
c at least one good step since restart then restart again
c
               do 23044 n = 1, num
                  p(n) = gm(n)
23044                continue
c
c initialize lambda_k to lambda_1
c
               xl = xlstart
c
c set lambda_k bar back to 0
c
               xlb = 0
               success = .true.
               delta = 1.0
               icount = 0
               fcount = 0
c
c return with error index if the failure count is > 2 and the
c number of iterations since the last restart is <= the failure count
c
            else
               ierr = 3
               iter = k
               goto 23009
            end if
         end if
      end if
c
c*************************************************************
c
c  ******** TEST FOR CONVERGENCE *****************
c
c*************************************************************
c
c
c adjust xl if delta < .25
c
      if (delta .lt. 0.25) then
               xl = 4.0*xl
      end if
```

```
c
c compute current rms error and the norm of the current descent
c direction
c
      rmserr = sqrt(error * 2)
      gsiz = snrm2(num, r, 1)
c
c if the error function can be reduced and the convergence counter
c is >= number of iterations before convergence check then reset
c convergence check counter and test for convergence.
c
      if((success .eq. .true.) .and. (ncount  .ge.  nfreq)) then
         ncount = 0
c
c Terminate if convergence too slow
c
         if(rmserr  .gt.  errdel * rmsold) then
            ierr = 3
            iter = k
            goto 23009
          end if
c
c save the current rms error for future check
c
          rmsold = rmserr
       end if
c
c Terminate when error satisfactory
c
       if(rmserr  .lt.  egoal) then
          ierr = 0
          iter = k
          goto 23009
       end if
c
c Terminate when gradient is too small
c
       if(gsiz  .lt.  gwgoal * max(1.0, wsiz)) then
          ierr = 2
          iter = k
          goto 23009
       end if
c
c*************************************************************
c
c  ******* GO BACK TO TOP OF MAIN LOOP ********
c
c*************************************************************
c
       goto 23008
c
c*************************************************************
c
c          ********** EXIT **********
c
c*************************************************************
c
c compute ratio of descent magnitude over weight magnitude
c
23009 continue
```

```
        gw = gsiz / wsiz
c
c rewind scratch file and write out graphics file
c
        ncol = 2
        rewind(fpscr)
        write(fpgrph,*) iter,ncol
        do 24000 i = 1,iter
           read(fpscr,*) it,rer
           write(fpgrph,*) it,rer
24000      continue
        return
        end
c*************************************************************
c Kolmogorov-Smirnov one-distribution test
c*************************************************************
      SUBROUTINE KSONE(DATA,N,D,PROB)
c*************************************************************
c
c From Press, et al. "Numerical Recipes"
c
c Given an array of N values, DATA, and given a user supplied
c function of a single variable, CUMUL, which is the
c cumulative distribution function ranging from 0 (for
c smallest values of its argument) to 1 (for largest values
c of its argument), this routine returns the K-S statistic, D,
c and the significance level PROB. Small values of PROB show
c that the cumulative distribution function of DATA is
c significantly different from CUMUL. The array DATA is
c modified by being sorted into ascending order.
c
c*************************************************************
      DIMENSION DATA(N)
      CALL SORT(N,DATA)
      EN=N
      D=0.
      FO=0.
      DO 11 J=1,N
        FN=J/EN
        FF=CUMUL(DATA(J))
        DT=AMAX1(ABS(FO-FF),ABS(FN-FF))
        IF(DT.GT.D)D=DT
        FO=FN
11    CONTINUE
      PROB=PROBKS(SQRT(EN)*D)
      RETURN
      END
c*************************************************************
      FUNCTION PROBKS(ALAM)
c*************************************************************
c
c From Press, et al., "Numerical Recipes"
c
c This function calculates the significance for the
c K-S statistic
c
c*************************************************************
      PARAMETER (EPS1=0.001, EPS2=1.E-8)
      A2=-2.*ALAM**2
      FAC=2.
```

```
      PROBKS=0.
      TERMBF=0.
      DO 11 J=1,100
        TERM=FAC*EXP(A2*J**2)
        PROBKS=PROBKS+TERM
        IF(ABS(TERM).LT.EPS1*TERMBF.OR.ABS(TERM).LT.EPS2*PROBKS)RETURN
        FAC=-FAC
        TERMBF=ABS(TERM)
11    CONTINUE
      PROBKS=1.
      RETURN
      END
c*************************************************************
      FUNCTION ERFCC(X)
c*************************************************************
c
c From Press, et al., "Numerical Recipes"
c
c Returns the complimentary error function with
c fractional error less than 1.2e-7
c
c*************************************************************
      Z=ABS(X)
      T=1./(1.+0.5*Z)
      ERFCC=T*EXP(-Z*Z-1.26551223+T*(1.00002368+T*(.37409196+
     *    T*(.09678418+T*(-.18628806+T*(.27886807+T*(-1.13520398+
     *    T*(1.48851587+T*(-.82215223+T*.17087277)))))))))
      IF (X.LT.0.) ERFCC=2.-ERFCC
      RETURN
      END
c*************************************************************
      SUBROUTINE SORT(N,RA)
c*************************************************************
c
c From Press, et al., "Numerical Recipes"
c
c Sorts an array, RA, of length N into ascending numerical
c order using the Heapsort algorithm. N is input; RA is
c replaced on output by its sorted rearrangement.
c
c*************************************************************
      DIMENSION RA(N)
      L=N/2+1
      IR=N
10    CONTINUE
        IF(L.GT.1)THEN
          L=L-1
          RRA=RA(L)
        ELSE
          RRA=RA(IR)
          RA(IR)=RA(1)
          IR=IR-1
          IF(IR.EQ.1)THEN
            RA(1)=RRA
            RETURN
          ENDIF
        ENDIF
        I=L
        J=L+L
20      IF(J.LE.IR)THEN
```

```
            IF(J.LT.IR)THEN
              IF(RA(J).LT.RA(J+1))J=J+1
            ENDIF
            IF(RRA.LT.RA(J))THEN
              RA(I)=RA(J)
              I=J
              J=J+J
            ELSE
              J=IR+1
            ENDIF
          GO TO 20
          ENDIF
          RA(I)=RRA
        GO TO 10
        END
c*************************************************************
      function erf(x)
c*************************************************************
c
c This function computes the error function
c
c*************************************************************
      erf = 1. - erfcc(x)
      return
      end
c*************************************************************
      function cumul(x)
c*************************************************************
c
c This function returns the cumulative distribution of
c the (0,1) normal distribution
c
c*************************************************************
      if (x .ge. 0.) then
            cumul = 0.5*(1.+erf(x/(sqrt(2.))))
      else
            cumul = 1. - 0.5*(1.+erf(-x/(sqrt(2.))))
      endif
      return
      end
```

APPENDIX B

SAMPLE OUTPUT FILE

NEURAL NET RUN REPORT

****

X

****

DATA SET

****

Training on xdspdown.trn

The number of data patterns is 397

****

NETWORK SPECIFICATION

****

INPUT NODES	HIDDEN NODES	OUTPUT NODES
37	60	1
WEIGHTS (I TO H)	**WEIGHTS (H TO O)**	**TOTAL WEIGHTS**
2280	61	2341

****

RUN PARAMETERS

****

```
 Random initial weights, seed      12345
 Regularization factor wfactor =    1.300E-03
1
 ********************************************************************************


                          RUN TERMINATION CRITERIA


 ********************************************************************************


 Maximum number of iterations is   1500
 Error checking frequency is every   1 iterations
 Run is terminated on either max iterations or
 one of the criteria below.
 RMS error <=    0.100E-02
 (RMS of g) <=    0.100E-11 * (RMS of w)
 (RMS err) >     1.00     * (RMS err    1 iterations ago)
 max iterations =   1500 # of weights =   2341
1
 ********************************************************************************


                               RUN RESULTS


 ********************************************************************************


                          R-SQ of nonlinear fit =   0.9961917

                          Pointwise  Error

            Mean                                Standard Deviation
        0.2527817E-05                               0.1701827E-03

           Minimum                                  Maximum
       -0.6710461E-03                              0.7603451E-03

      Data        Output        Desired        Predicted        Absolute
      Index       Node          Output         Output           Error
        1           1        -0.1163E-02    -0.1325E-02       0.1616E-03
        2           1        -0.3243E-02    -0.3874E-02       0.6303E-03
        3           1        -0.7414E-02    -0.7286E-02      -0.1279E-03
        4           1        -0.9268E-02    -0.9583E-02       0.3158E-03
        5           1        -0.1280E-01    -0.1326E-01       0.4662E-03
```

6	1	-0.1458E-01	-0.1474E-01	0.1646E-03
7	1	-0.1706E-01	-0.1717E-01	0.1092E-03
8	1	-0.1833E-01	-0.1842E-01	0.8106E-04
9	1	-0.2032E-01	-0.2079E-01	0.4792E-03
10	1	-0.2156E-01	-0.2180E-01	0.2378E-03
11	1	-0.2311E-01	-0.2283E-01	-0.2781E-03
12	1	-0.2318E-01	-0.2311E-01	-0.7785E-04
13	1	-0.4257E-02	-0.3586E-02	-0.6710E-03
14	1	-0.6372E-02	-0.6389E-02	0.1641E-04
15	1	-0.1008E-01	-0.9664E-02	-0.4157E-03
16	1	-0.1155E-01	-0.1149E-01	-0.6624E-04
17	1	-0.1518E-01	-0.1489E-01	-0.2914E-03
18	1	-0.1655E-01	-0.1609E-01	-0.4650E-03
19	1	-0.1884E-01	-0.1832E-01	-0.5189E-03
20	1	-0.1973E-01	-0.1938E-01	-0.3496E-03
21	1	-0.2196E-01	-0.2135E-01	-0.6079E-03
22	1	-0.2282E-01	-0.2230E-01	-0.5200E-03
23	1	-0.2343E-01	-0.2282E-01	-0.6117E-03
24	1	-0.1175E-02	-0.8425E-03	-0.3325E-03
25	1	-0.5328E-02	-0.5506E-02	0.1773E-03
26	1	-0.7181E-02	-0.6983E-02	-0.1988E-03
27	1	-0.1040E-01	-0.1081E-01	0.4007E-03
28	1	-0.1228E-01	-0.1273E-01	0.4491E-03
29	1	-0.1525E-01	-0.1556E-01	0.3107E-03
30	1	-0.1632E-01	-0.1659E-01	0.2763E-03
31	1	-0.1830E-01	-0.1844E-01	0.1442E-03
32	1	-0.1923E-01	-0.1932E-01	0.8399E-04
33	1	-0.2129E-01	-0.2151E-01	0.2196E-03
34	1	-0.2179E-01	-0.2187E-01	0.7486E-04
35	1	0.1793E-05	-0.2798E-03	0.2816E-03
36	1	-0.2097E-02	-0.2078E-02	-0.1882E-04
37	1	-0.6162E-02	-0.5910E-02	-0.2519E-03
38	1	-0.7888E-02	-0.7834E-02	-0.5306E-04
39	1	-0.1129E-01	-0.1145E-01	0.1600E-03
40	1	-0.1296E-01	-0.1312E-01	0.1557E-03
41	1	-0.1534E-01	-0.1548E-01	0.1386E-03
42	1	-0.1643E-01	-0.1645E-01	0.2010E-04
43	1	-0.1831E-01	-0.1826E-01	-0.5399E-04
44	1	-0.1953E-01	-0.1919E-01	-0.3315E-03
45	1	-0.2092E-01	-0.2113E-01	0.2115E-03
46	1	-0.2094E-01	-0.2108E-01	0.1425E-03
47	1	-0.1468E-02	-0.1398E-02	-0.6996E-04
48	1	-0.3597E-02	-0.3544E-02	-0.5235E-04
49	1	-0.7325E-02	-0.6886E-02	-0.4393E-03
50	1	-0.8745E-02	-0.8568E-02	-0.1776E-03
51	1	-0.1230E-01	-0.1223E-01	-0.6508E-04
52	1	-0.1367E-01	-0.1364E-01	-0.2880E-04
53	1	-0.1584E-01	-0.1581E-01	-0.3006E-04
54	1	-0.1671E-01	-0.1678E-01	0.7707E-04
55	1	-0.1888E-01	-0.1860E-01	-0.2891E-03
56	1	-0.1975E-01	-0.2014E-01	0.3973E-03
57	1	-0.2030E-01	-0.2040E-01	0.1018E-03
58	1	0.2488E-02	0.1828E-02	0.6604E-03
59	1	-0.1376E-02	-0.1298E-02	-0.7775E-04
60	1	-0.3241E-02	-0.3020E-02	-0.2215E-03
61	1	-0.6488E-02	-0.6362E-02	-0.1268E-03
62	1	-0.8302E-02	-0.8139E-02	-0.1638E-03
63	1	-0.1122E-01	-0.1091E-01	-0.3138E-03
64	1	-0.1219E-01	-0.1208E-01	-0.1158E-03
65	1	-0.1420E-01	-0.1424E-01	0.4655E-04

66	1	-0.1507E-01	-0.1533E-01	0.2595E-03
67	1	-0.1700E-01	-0.1699E-01	-0.4636E-05
68	1	-0.1746E-01	-0.1752E-01	0.5755E-04
69	1	0.2593E-02	0.1833E-02	0.7603E-03
70	1	0.6198E-03	0.2878E-03	0.3320E-03
71	1	-0.3264E-02	-0.3036E-02	-0.2276E-03
72	1	-0.4950E-02	-0.4917E-02	-0.3323E-04
73	1	-0.8222E-02	-0.8189E-02	-0.3262E-04
74	1	-0.9858E-02	-0.9702E-02	-0.1557E-03
75	1	-0.1209E-01	-0.1207E-01	-0.1951E-04
76	1	-0.1316E-01	-0.1311E-01	-0.5524E-04
77	1	-0.1494E-01	-0.1525E-01	0.3142E-03
78	1	-0.1612E-01	-0.1623E-01	0.1101E-03
79	1	-0.1741E-01	-0.1735E-01	-0.5937E-04
80	1	-0.1726E-01	-0.1721E-01	-0.4983E-04
81	1	0.6733E-03	0.2910E-03	0.3823E-03
82	1	-0.1272E-02	-0.1278E-02	0.6169E-05
83	1	-0.4924E-02	-0.4856E-02	-0.6874E-04
84	1	-0.6306E-02	-0.6314E-02	0.8279E-05
85	1	-0.9831E-02	-0.9574E-02	-0.2563E-03
86	1	-0.1111E-01	-0.1078E-01	-0.3309E-03
87	1	-0.1316E-01	-0.1297E-01	-0.1877E-03
88	1	-0.1406E-01	-0.1405E-01	-0.1310E-04
89	1	-0.1601E-01	-0.1611E-01	0.9975E-04
90	1	-0.1678E-01	-0.1678E-01	-0.1146E-05
91	1	-0.1721E-01	-0.1714E-01	-0.7141E-04
92	1	0.2579E-02	0.1851E-02	0.7282E-03
93	1	-0.1295E-02	-0.1287E-02	-0.8701E-05
94	1	-0.3260E-02	-0.3002E-02	-0.2582E-03
95	1	-0.6250E-02	-0.6342E-02	0.9177E-04
96	1	-0.8164E-02	-0.8125E-02	-0.3906E-04
97	1	-0.1105E-01	-0.1082E-01	-0.2297E-03
98	1	-0.1205E-01	-0.1198E-01	-0.7428E-04
99	1	-0.1400E-01	-0.1406E-01	0.6104E-04
100	1	-0.1487E-01	-0.1512E-01	0.2482E-03
101	1	-0.1679E-01	-0.1675E-01	-0.4665E-04
102	1	-0.1721E-01	-0.1710E-01	-0.1147E-03
103	1	0.2526E-02	0.1836E-02	0.6905E-03
104	1	0.6611E-03	0.2722E-03	0.3889E-03
105	1	-0.3156E-02	-0.3043E-02	-0.1126E-03
106	1	-0.4917E-02	-0.4940E-02	0.2303E-04
107	1	-0.8160E-02	-0.8220E-02	0.5995E-04
108	1	-0.9743E-02	-0.9713E-02	-0.3018E-04
109	1	-0.1205E-01	-0.1208E-01	0.3676E-04
110	1	-0.1307E-01	-0.1310E-01	0.2860E-04
111	1	-0.1482E-01	-0.1519E-01	0.3719E-03
112	1	-0.1594E-01	-0.1615E-01	0.2065E-03
113	1	-0.1720E-01	-0.1710E-01	-0.9839E-04
114	1	-0.1711E-01	-0.1714E-01	0.2684E-04
115	1	0.5598E-03	0.3568E-03	0.2031E-03
116	1	-0.1313E-02	-0.1199E-02	-0.1141E-03
117	1	-0.4561E-02	-0.4660E-02	0.9950E-04
118	1	-0.5753E-02	-0.5913E-02	0.1608E-03
119	1	-0.9051E-02	-0.9135E-02	0.8416E-04
120	1	-0.1022E-01	-0.1014E-01	-0.7277E-04
121	1	-0.1197E-01	-0.1196E-01	-0.8461E-05
122	1	-0.1273E-01	-0.1267E-01	-0.5241E-04
123	1	-0.1443E-01	-0.1409E-01	-0.3391E-03
124	1	-0.1503E-01	-0.1475E-01	-0.2820E-03
125	1	-0.1505E-01	-0.1485E-01	-0.2073E-03

126	1	0.2346E-02	0.1974E-02	0.3718E-03
127	1	-0.1313E-02	-0.1228E-02	-0.8421E-04
128	1	-0.3114E-02	-0.2908E-02	-0.2066E-03
129	1	-0.5853E-02	-0.5957E-02	0.1037E-03
130	1	-0.7579E-02	-0.7659E-02	0.8058E-04
131	1	-0.1019E-01	-0.1023E-01	0.3708E-04
132	1	-0.1105E-01	-0.1120E-01	0.1499E-03
133	1	-0.1275E-01	-0.1276E-01	0.1070E-04
134	1	-0.1346E-01	-0.1347E-01	0.1532E-04
135	1	-0.1500E-01	-0.1486E-01	-0.1390E-03
136	1	-0.1533E-01	-0.1523E-01	-0.9902E-04
137	1	0.2332E-02	0.1966E-02	0.3655E-03
138	1	0.5386E-03	0.3271E-03	0.2115E-03
139	1	-0.3086E-02	-0.2893E-02	-0.1924E-03
140	1	-0.4683E-02	-0.4645E-02	-0.3774E-04
141	1	-0.7598E-02	-0.7578E-02	-0.2066E-04
142	1	-0.9145E-02	-0.9116E-02	-0.2846E-04
143	1	-0.1107E-01	-0.1111E-01	0.4086E-04
144	1	-0.1198E-01	-0.1195E-01	-0.2667E-04
145	1	-0.1342E-01	-0.1340E-01	-0.2403E-04
146	1	-0.1441E-01	-0.1412E-01	-0.2910E-03
147	1	-0.1534E-01	-0.1514E-01	-0.1986E-03
148	1	-0.1516E-01	-0.1493E-01	-0.2296E-03
149	1	0.4563E-03	0.3172E-03	0.1391E-03
150	1	-0.1461E-02	-0.1251E-02	-0.2097E-03
151	1	-0.4604E-02	-0.4646E-02	0.4264E-04
152	1	-0.5921E-02	-0.5879E-02	-0.4141E-04
153	1	-0.9140E-02	-0.9110E-02	-0.3020E-04
154	1	-0.1020E-01	-0.1015E-01	-0.5578E-04
155	1	-0.1197E-01	-0.1197E-01	-0.6506E-05
156	1	-0.1271E-01	-0.1270E-01	-0.8681E-05
157	1	-0.1445E-01	-0.1416E-01	-0.2956E-03
158	1	-0.1509E-01	-0.1485E-01	-0.2387E-03
159	1	-0.1510E-01	-0.1502E-01	-0.8416E-04
160	1	0.2195E-02	0.1937E-02	0.2587E-03
161	1	-0.1461E-02	-0.1286E-02	-0.1746E-03
162	1	-0.3210E-02	-0.2960E-02	-0.2495E-03
163	1	-0.5946E-02	-0.5935E-02	-0.1170E-04
164	1	-0.7696E-02	-0.7641E-02	-0.5500E-04
165	1	-0.1025E-01	-0.1020E-01	-0.5539E-04
166	1	-0.1109E-01	-0.1117E-01	0.7675E-04
167	1	-0.1273E-01	-0.1273E-01	0.1186E-05
168	1	-0.1342E-01	-0.1346E-01	0.4102E-04
169	1	-0.1509E-01	-0.1486E-01	-0.2286E-03
170	1	-0.1536E-01	-0.1518E-01	-0.1824E-03
171	1	0.1812E-02	0.1528E-02	0.2837E-03
172	1	0.7500E-04	0.7359E-05	0.6764E-04
173	1	-0.3505E-02	-0.3363E-02	-0.1422E-03
174	1	-0.4998E-02	-0.5042E-02	0.4353E-04
175	1	-0.7933E-02	-0.7867E-02	-0.6615E-04
176	1	-0.9400E-02	-0.9189E-02	-0.2113E-03
177	1	-0.1137E-01	-0.1145E-01	0.7965E-04
178	1	-0.1225E-01	-0.1229E-01	0.3176E-04
179	1	-0.1361E-01	-0.1378E-01	0.1710E-03
180	1	-0.1457E-01	-0.1455E-01	-0.2769E-04
181	1	-0.1543E-01	-0.1532E-01	-0.1077E-03
182	1	-0.1522E-01	-0.1547E-01	0.2576E-03
183	1	0.1029E-03	-0.1759E-04	0.1205E-03
184	1	-0.1758E-02	-0.1682E-02	-0.7618E-04
185	1	-0.4922E-02	-0.5030E-02	0.1086E-03

186	1	-0.6161E-02	-0.6177E-02	0.1650E-04
187	1	-0.9299E-02	-0.9189E-02	-0.1096E-03
188	1	-0.1046E-01	-0.1050E-01	0.3537E-04
189	1	-0.1210E-01	-0.1228E-01	0.1823E-03
190	1	-0.1281E-01	-0.1302E-01	0.2124E-03
191	1	-0.1452E-01	-0.1452E-01	-0.5068E-05
192	1	-0.1510E-01	-0.1517E-01	0.6611E-04
193	1	-0.1517E-01	-0.1540E-01	0.2394E-03
194	1	0.1814E-02	0.1413E-02	0.4013E-03
195	1	-0.1784E-02	-0.1712E-02	-0.7168E-04
196	1	-0.3378E-02	-0.3404E-02	0.2578E-04
197	1	-0.6211E-02	-0.6193E-02	-0.1866E-04
198	1	-0.7857E-02	-0.7868E-02	0.1148E-04
199	1	-0.1041E-01	-0.1049E-01	0.7709E-04
200	1	-0.1124E-01	-0.1144E-01	0.1959E-03
201	1	-0.1281E-01	-0.1299E-01	0.1845E-03
202	1	-0.1349E-01	-0.1373E-01	0.2404E-03
203	1	-0.1510E-01	-0.1508E-01	-0.2208E-04
204	1	-0.1532E-01	-0.1505E-01	-0.2702E-03
205	1	0.1784E-02	0.1357E-02	0.4272E-03
206	1	-0.8735E-05	-0.9434E-04	0.8561E-04
207	1	-0.3454E-02	-0.3441E-02	-0.1247E-04
208	1	-0.5023E-02	-0.5096E-02	0.7303E-04
209	1	-0.7906E-02	-0.7897E-02	-0.9729E-05
210	1	-0.9323E-02	-0.9193E-02	-0.1293E-03
211	1	-0.1127E-01	-0.1146E-01	0.1944E-03
212	1	-0.1217E-01	-0.1228E-01	0.1033E-03
213	1	-0.1353E-01	-0.1373E-01	0.1951E-03
214	1	-0.1457E-01	-0.1446E-01	-0.1102E-03
215	1	-0.1534E-01	-0.1503E-01	-0.3132E-03
216	1	-0.1518E-01	-0.1525E-01	0.7089E-04
217	1	-0.4026E-04	-0.1137E-03	0.7345E-04
218	1	-0.1908E-02	-0.1799E-02	-0.1091E-03
219	1	-0.5096E-02	-0.5142E-02	0.4644E-04
220	1	-0.6308E-02	-0.6257E-02	-0.5129E-04
221	1	-0.9394E-02	-0.9231E-02	-0.1633E-03
222	1	-0.1048E-01	-0.1056E-01	0.7619E-04
223	1	-0.1219E-01	-0.1233E-01	0.1323E-03
224	1	-0.1285E-01	-0.1305E-01	0.2060E-03
225	1	-0.1454E-01	-0.1453E-01	-0.4727E-05
226	1	-0.1506E-01	-0.1515E-01	0.8687E-04
227	1	-0.1518E-01	-0.1536E-01	0.1812E-03
228	1	0.1311E-02	0.1654E-02	-0.3427E-03
229	1	-0.1891E-02	-0.2049E-02	0.1577E-03
230	1	-0.3521E-02	-0.3429E-02	-0.9240E-04
231	1	-0.5892E-02	-0.6012E-02	0.1197E-03
232	1	-0.7497E-02	-0.7371E-02	-0.1265E-03
233	1	-0.9741E-02	-0.9818E-02	0.7623E-04
234	1	-0.1046E-01	-0.1033E-01	-0.1242E-03
235	1	-0.1186E-01	-0.1177E-01	-0.9780E-04
236	1	-0.1243E-01	-0.1236E-01	-0.6552E-04
237	1	-0.1378E-01	-0.1358E-01	-0.1990E-03
238	1	-0.1391E-01	-0.1411E-01	0.1992E-03
239	1	0.1502E-02	0.1626E-02	-0.1247E-03
240	1	-0.2200E-03	-0.4421E-03	0.2222E-03
241	1	-0.3546E-02	-0.3500E-02	-0.4610E-04
242	1	-0.4872E-02	-0.4883E-02	0.1080E-04
243	1	-0.7472E-02	-0.7472E-02	0.1611E-06
244	1	-0.8797E-02	-0.8870E-02	0.7331E-04
245	1	-0.1048E-01	-0.1041E-01	-0.7584E-04

246	1	-0.1125E-01	-0.1121E-01	-0.4116E-04
247	1	-0.1240E-01	-0.1243E-01	0.2545E-04
248	1	-0.1330E-01	-0.1305E-01	-0.2491E-03
249	1	-0.1394E-01	-0.1420E-01	0.2620E-03
250	1	-0.1359E-01	-0.1424E-01	0.6536E-03
251	1	-0.1996E-03	-0.5163E-03	0.3167E-03
252	1	-0.1992E-02	-0.2231E-02	0.2390E-03
253	1	-0.4872E-02	-0.5018E-02	0.1458E-03
254	1	-0.5994E-02	-0.6229E-02	0.2346E-03
255	1	-0.8822E-02	-0.8998E-02	0.1755E-03
256	1	-0.9792E-02	-0.1002E-01	0.2262E-03
257	1	-0.1127E-01	-0.1132E-01	0.4582E-04
258	1	-0.1181E-01	-0.1193E-01	0.1196E-03
259	1	-0.1332E-01	-0.1314E-01	-0.1788E-03
260	1	-0.1378E-01	-0.1376E-01	-0.2162E-04
261	1	-0.1359E-01	-0.1432E-01	0.7344E-03
262	1	0.1370E-02	0.1545E-02	-0.1753E-03
263	1	-0.1967E-02	-0.2261E-02	0.2944E-03
264	1	-0.3496E-02	-0.3711E-02	0.2152E-03
265	1	-0.6019E-02	-0.6258E-02	0.2388E-03
266	1	-0.7573E-02	-0.7647E-02	0.7415E-04
267	1	-0.9843E-02	-0.1006E-01	0.2138E-03
268	1	-0.1051E-01	-0.1057E-01	0.6175E-04
269	1	-0.1184E-01	-0.1198E-01	0.1414E-03
270	1	-0.1240E-01	-0.1258E-01	0.1750E-03
271	1	-0.1381E-01	-0.1382E-01	0.1284E-04
272	1	-0.1399E-01	-0.1435E-01	0.3623E-03
273	1	0.1428E-02	0.1494E-02	-0.6550E-04
274	1	-0.1768E-03	-0.6132E-03	0.4364E-03
275	1	-0.3546E-02	-0.3842E-02	0.2959E-03
276	1	-0.4948E-02	-0.5141E-02	0.1929E-03
277	1	-0.7472E-02	-0.7792E-02	0.3206E-03
278	1	-0.8899E-02	-0.9173E-02	0.2744E-03
279	1	-0.1058E-01	-0.1073E-01	0.1504E-03
280	1	-0.1125E-01	-0.1153E-01	0.2811E-03
281	1	-0.1245E-01	-0.1273E-01	0.2828E-03
282	1	-0.1337E-01	-0.1335E-01	-0.2060E-04
283	1	-0.1404E-01	-0.1448E-01	0.4380E-03
284	1	-0.1369E-01	-0.1442E-01	0.7331E-03
285	1	-0.1647E-02	-0.1170E-02	-0.4774E-03
286	1	-0.3392E-02	-0.2985E-02	-0.4070E-03
287	1	-0.6372E-02	-0.6067E-02	-0.3054E-03
288	1	-0.7418E-02	-0.7245E-02	-0.1732E-03
289	1	-0.1027E-01	-0.1006E-01	-0.2093E-03
290	1	-0.1134E-01	-0.1114E-01	-0.2041E-03
291	1	-0.1282E-01	-0.1250E-01	-0.3253E-03
292	1	-0.1339E-01	-0.1309E-01	-0.2974E-03
293	1	-0.1487E-01	-0.1433E-01	-0.5428E-03
294	1	-0.1533E-01	-0.1495E-01	-0.3819E-03
295	1	-0.1526E-01	-0.1498E-01	-0.2808E-03
296	1	0.1156E-03	0.6177E-03	-0.5021E-03
297	1	-0.3390E-02	-0.3001E-02	-0.3885E-03
298	1	-0.4994E-02	-0.4828E-02	-0.1658E-03
299	1	-0.7465E-02	-0.7257E-02	-0.2080E-03
300	1	-0.9094E-02	-0.8710E-02	-0.3845E-03
301	1	-0.1139E-01	-0.1116E-01	-0.2314E-03
302	1	-0.1210E-01	-0.1176E-01	-0.3431E-03
303	1	-0.1348E-01	-0.1312E-01	-0.3610E-03
304	1	-0.1404E-01	-0.1373E-01	-0.3195E-03
305	1	-0.1540E-01	-0.1497E-01	-0.4250E-03

306	1	-0.1561E-01	-0.1534E-01	-0.2688E-03
307	1	0.2924E-04	0.5461E-03	-0.5169E-03
308	1	-0.1732E-02	-0.1244E-02	-0.4885E-03
309	1	-0.5044E-02	-0.4905E-02	-0.1390E-03
310	1	-0.6420E-02	-0.6154E-02	-0.2657E-03
311	1	-0.9094E-02	-0.8825E-02	-0.2697E-03
312	1	-0.1047E-01	-0.1020E-01	-0.2719E-03
313	1	-0.1210E-01	-0.1190E-01	-0.2039E-03
314	1	-0.1289E-01	-0.1267E-01	-0.2269E-03
315	1	-0.1407E-01	-0.1386E-01	-0.2134E-03
316	1	-0.1489E-01	-0.1448E-01	-0.4059E-03
317	1	-0.1561E-01	-0.1541E-01	-0.1974E-03
318	1	-0.1536E-01	-0.1500E-01	-0.3505E-03
319	1	-0.1768E-02	-0.1296E-02	-0.4721E-03
320	1	-0.3517E-02	-0.3096E-02	-0.4210E-03
321	1	-0.6471E-02	-0.6221E-02	-0.2501E-03
322	1	-0.7517E-02	-0.7402E-02	-0.1148E-03
323	1	-0.1047E-01	-0.1025E-01	-0.2253E-03
324	1	-0.1144E-01	-0.1133E-01	-0.1119E-03
325	1	-0.1287E-01	-0.1272E-01	-0.1541E-03
326	1	-0.1346E-01	-0.1330E-01	-0.1557E-03
327	1	-0.1492E-01	-0.1452E-01	-0.3948E-03
328	1	-0.1543E-01	-0.1512E-01	-0.3129E-03
329	1	-0.1536E-01	-0.1500E-01	-0.3622E-03
330	1	-0.1417E-03	0.4591E-03	-0.6008E-03
331	1	-0.3443E-02	-0.3127E-02	-0.3155E-03
332	1	-0.5073E-02	-0.4982E-02	-0.9076E-04
333	1	-0.7597E-02	-0.7437E-02	-0.1592E-03
334	1	-0.9151E-02	-0.8901E-02	-0.2497E-03
335	1	-0.1147E-01	-0.1137E-01	-0.1056E-03
336	1	-0.1216E-01	-0.1200E-01	-0.1641E-03
337	1	-0.1354E-01	-0.1336E-01	-0.1854E-03
338	1	-0.1408E-01	-0.1396E-01	-0.1226E-03
339	1	-0.1544E-01	-0.1517E-01	-0.2626E-03
340	1	-0.1569E-01	-0.1545E-01	-0.2436E-03
341	1	0.7671E-04	0.2077E-03	-0.1310E-03
342	1	-0.1559E-02	-0.1531E-02	-0.2731E-04
343	1	-0.4895E-02	-0.5171E-02	0.2755E-03
344	1	-0.6271E-02	-0.6485E-02	0.2134E-03
345	1	-0.9049E-02	-0.9072E-02	0.2285E-04
346	1	-0.1035E-01	-0.1045E-01	0.9730E-04
347	1	-0.1206E-01	-0.1218E-01	0.1238E-03
348	1	-0.1283E-01	-0.1294E-01	0.1102E-03
349	1	-0.1400E-01	-0.1417E-01	0.1691E-03
350	1	-0.1490E-01	-0.1484E-01	-0.5521E-04
351	1	-0.1564E-01	-0.1564E-01	0.1378E-06
352	1	-0.1532E-01	-0.1552E-01	0.2044E-03
353	1	-0.1442E-02	-0.1548E-02	0.1067E-03
354	1	-0.3239E-02	-0.3317E-02	0.7767E-04
355	1	-0.6271E-02	-0.6495E-02	0.2233E-03
356	1	-0.7444E-02	-0.7616E-02	0.1714E-03
357	1	-0.1035E-01	-0.1046E-01	0.1144E-03
358	1	-0.1137E-01	-0.1151E-01	0.1408E-03
359	1	-0.1278E-01	-0.1297E-01	0.1897E-03
360	1	-0.1342E-01	-0.1356E-01	0.1477E-03
361	1	-0.1490E-01	-0.1484E-01	-0.5431E-04
362	1	-0.1538E-01	-0.1544E-01	0.5332E-04
363	1	-0.1534E-01	-0.1542E-01	0.7625E-04
364	1	0.1402E-03	0.1743E-03	-0.3406E-04
365	1	-0.3367E-02	-0.3362E-02	-0.4798E-05

```
        366           1      -0.4870E-02  -0.5203E-02         0.3329E-03
        367           1      -0.7444E-02  -0.7655E-02         0.2108E-03
        368           1      -0.9024E-02  -0.9137E-02         0.1137E-03
        369           1      -0.1137E-01  -0.1156E-01         0.1853E-03
        370           1      -0.1203E-01  -0.1226E-01         0.2285E-03
        371           1      -0.1344E-01  -0.1362E-01         0.1755E-03
        372           1      -0.1403E-01  -0.1424E-01         0.2143E-03
        373           1      -0.1541E-01  -0.1550E-01         0.8990E-04
        374           1      -0.1567E-01  -0.1562E-01        -0.5090E-04
        375           1       0.1656E-03   0.1723E-03        -0.6694E-05
        376           1      -0.1515E-02  -0.1587E-02         0.7160E-04
        377           1      -0.4870E-02  -0.5215E-02         0.3451E-03
        378           1      -0.6322E-02  -0.6539E-02         0.2165E-03
        379           1      -0.8948E-02  -0.9163E-02         0.2151E-03
        380           1      -0.1032E-01  -0.1055E-01         0.2268E-03
        381           1      -0.1206E-01  -0.1232E-01         0.2619E-03
        382           1      -0.1283E-01  -0.1308E-01         0.2531E-03
        383           1      -0.1403E-01  -0.1431E-01         0.2854E-03
        384           1      -0.1490E-01  -0.1497E-01         0.7639E-04
        385           1      -0.1564E-01  -0.1571E-01         0.6475E-04
        386           1      -0.1537E-01  -0.1564E-01         0.2691E-03
        387           1      -0.1531E-02  -0.1609E-02         0.7815E-04
        388           1      -0.3367E-02  -0.3397E-02         0.3005E-04
        389           1      -0.6373E-02  -0.6587E-02         0.2146E-03
        390           1      -0.7444E-02  -0.7718E-02         0.2736E-03
        391           1      -0.1035E-01  -0.1061E-01         0.2599E-03
        392           1      -0.1132E-01  -0.1167E-01         0.3484E-03
        393           1      -0.1283E-01  -0.1316E-01         0.3296E-03
        394           1      -0.1339E-01  -0.1376E-01         0.3719E-03
        395           1      -0.1487E-01  -0.1505E-01         0.1775E-03
        396           1      -0.1538E-01  -0.1565E-01         0.2622E-03
        397           1      -0.1537E-01  -0.1571E-01         0.3394E-03
Kolmogorov-Smirnov distance statistic =   0.9152800E-01
Kolmogorov-Smirnov significance value =   0.2583816E-02
Iter  1500; ierr 1 : iteration limit
Used 1500 iterations;   3011 function calls; Err  0.000; |g|/|w| 1.018E-04
Rms change in weights 0.070
Weights written to file xdspdntrnwt.out
```